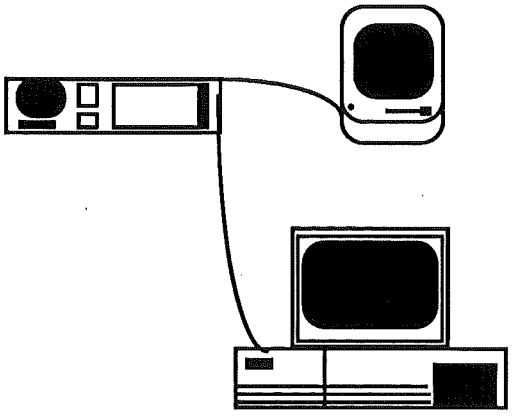
X.25 Protocol Analyser

Honours Project Report

C.S.Daniell 1989



Supervisor: Ray Hunt

Department of Computer Science
University of Canterbury

Contents

1 Introduction

The aim of this project was to produce a tool to aid with the statistical analysis of the data flow on communications lines operating with the X.25 packet switching protocol. The only tool available at the University of Canterbury which gave access to the traffic in a usable form was the Comstate 1 Datascope [1]. The level of data analysis provided by the Comstate 1 Datascope is minimal. It provides data displays at byte, frame and packet levels but provides no form of numerical overview of these results. This means that if statistical analysis of packet distributions and lengths etc is required, then obtaining these figures involves a manual count of them on the screen. This is a very time consuming and inaccurate process, but it is the only available way of obtaining useful summary information from the Comstate 1.

To reduce the time and effort involved in this sort of analysis the aim was to provide some personal computer based software to analyse the data captured by the Comstate 1. The software has been designed to provide a summary of the contents of the captured sample and present them in a numerical and graphical form. Statistics, which are calculated from this summary, such as percentage of time that a line is idle, average data packet length and the overhead the protocol adds to the transmission, are probably the most useful.

An aim in the development of the software tool was to provide statistics for each logical channel as well as loads for the total link. The major need for this type of information is in the evaluation of the loads and utilization of the link. This form of analysis is needed to determine if the link is actually the bottle-neck in a communications network. It is often singled out as the cause of the problem but it is not usually possible to quantify the percentage of the link capacity actually used or view the time distribution of its usage.

2 Technical

2.1 X.25 Frame and Packet Header Format

The frame level (layer 2) header consists of two octets. The first octet is the address octet. The information contained in this octet is not required for the analysis being preformed so this field is ignored except for the purpose of adding to the count of control characters. The second is the control octet. Bit 1 (least significant) of this octet contains a '0' for frames containing information for higher levels and a '1' for all other frames. All other frames only contain layer 2 control information. Table 1 contains the format of the layer 2 control octet (from [2]).

Frame Type	Frame header (Octet 2)							
	8	7	6	5	4	3	2	1
RR Packet		N(R)		P/F	0	0	0	1
RNR Packet		N(R)		P/F	0	1	0	1
REJ Packet		N(R)		P/F	1	0	0	1
Data Packet		N(R)		P		N(S)		0
All The Others	*	*	*	*	*	*	*	1

Table 1 Frame Headers

The layer 3 header consists of three octets: The upper half of the first octet contains the qualifier bit, '1' in the case of a data packet, '0' for all other packets, and then the sequence '001'. No information is needed from these bits. The lower half of the first octet combined with the second octet makes up the logical channel number. The third octet is the packet type identifier. Table 2 contains the format of the layer 3 packet type identifier octet (from [2]).

Packet Type	Packet header (Octet 3)							
	8	7	6	5	4	3	2	1
RR Packet		P(R)		0	0	0	0	1
RNR Packet		P(R)		0	0	1	0	1
Data Packet		P(R)		M		P(S)		0
All The Others	*	*	*	*	*	*	*	1

Table 2 Packet Headers

The packets with a '0' as the least significant bit of the third octet are packets containing user data. All other packets contain only layer 3 supervisory information.

Figure 1 below shows how the layer two and three packet headers match on to a data sample. The file transfered from the Datascope has the flags suppressed (before address octet and after frame check sequence). The hexidecimal values corresponding to the mnemonics in the data sample are in table 3.

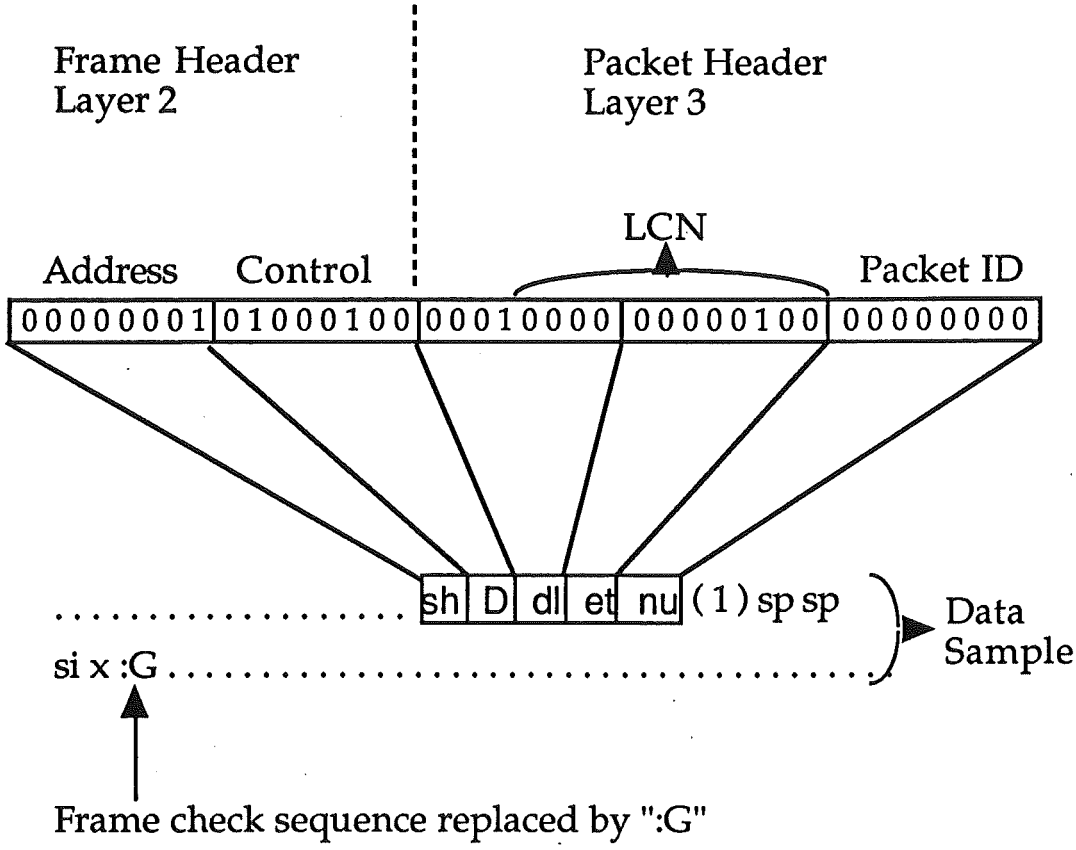


Figure 1 Data sample from Comstate 1 to Frame and Packet Headers

2.2 The Hardware

The hardware equipment used in this project is as follows:

Comstate 1 Datascope : The reason for the use of the Comstate 1 Datascope in this project is to obtain traffic samples of an X.25 connection. It was the only piece of hardware available at the University of Canterbury capable of doing this without writing a serial driver for a computer. That was considered outside the scope of this project.

The Comstate 1 Datascope does not have the ability to read data off the line it is monitoring and write the data out again to one of its other ports in real-time. To write it out again it has to be in freeze mode which means it is no longer monitoring the line. The ability to monitor and write out data at the same time is a feature that appears well within the capabilities of the hardware of the Comstate 1 Datascope but is not an option the designers made available.

For several reasons the Comstate 1 Datascope limits the size of sample it is possible to obtain. The buffer size of the Comstate 1 Datascope is limited to 64 kbytes, of which 32 kbytes is devoted to data characters from the line. This represents a data sample of 13.33 seconds on a fully utilized full duplex line at 9.6 kbps. A datascope with a floppy disc drive or internal hard disc would have been most useful, but was unfortunately not available.

Apple Macintosh Plus : The main reason for development of the project on the Macintosh was the availability of hardware, software and technical support. As the use of Macintosh hardware in the commercial environment is not as wide spread as in the academic environment, development was not limited to just the Macintosh.

PC (Exzel AT 286 IBM PC compatible) : Development was done in parallel on a Dos machine because of the wide spread availability of PC's in the commercial environment. It was felt that if the analyzer was to be useful it had to be available in a convenient form.

Cable : A cable for connecting the printer port on the back of the Comstate 1 Datascope to the modem port on the Macintosh for file transfer was not available. The cable shown in Figure 2 was constructed for this purpose. For file transfer from the Comstate 1 Datascope to the PC the standard RS232 cable which comes with the Datascope is all that is required to connect the printer port of the Datascope to Coms port 1 of the PC.

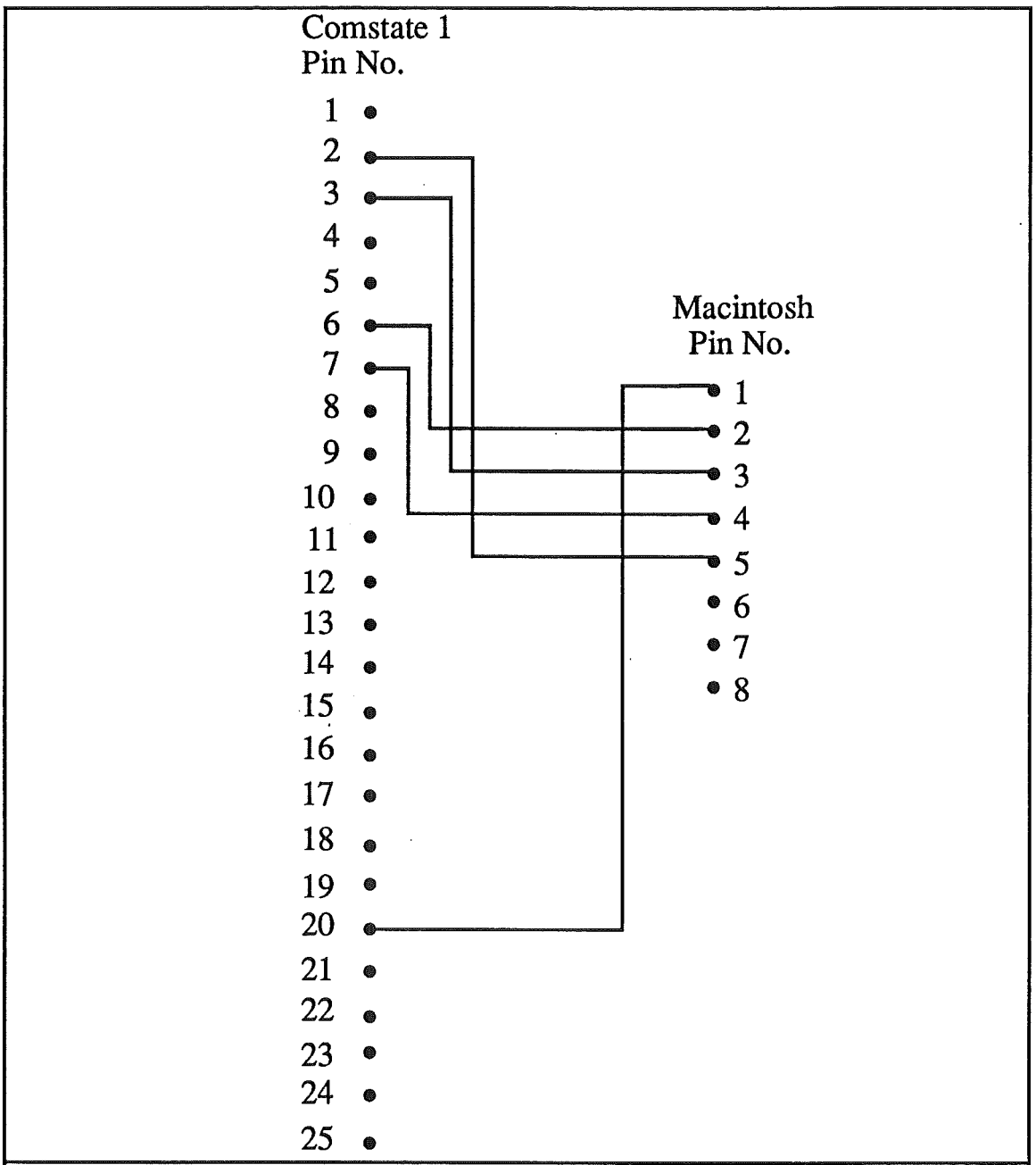
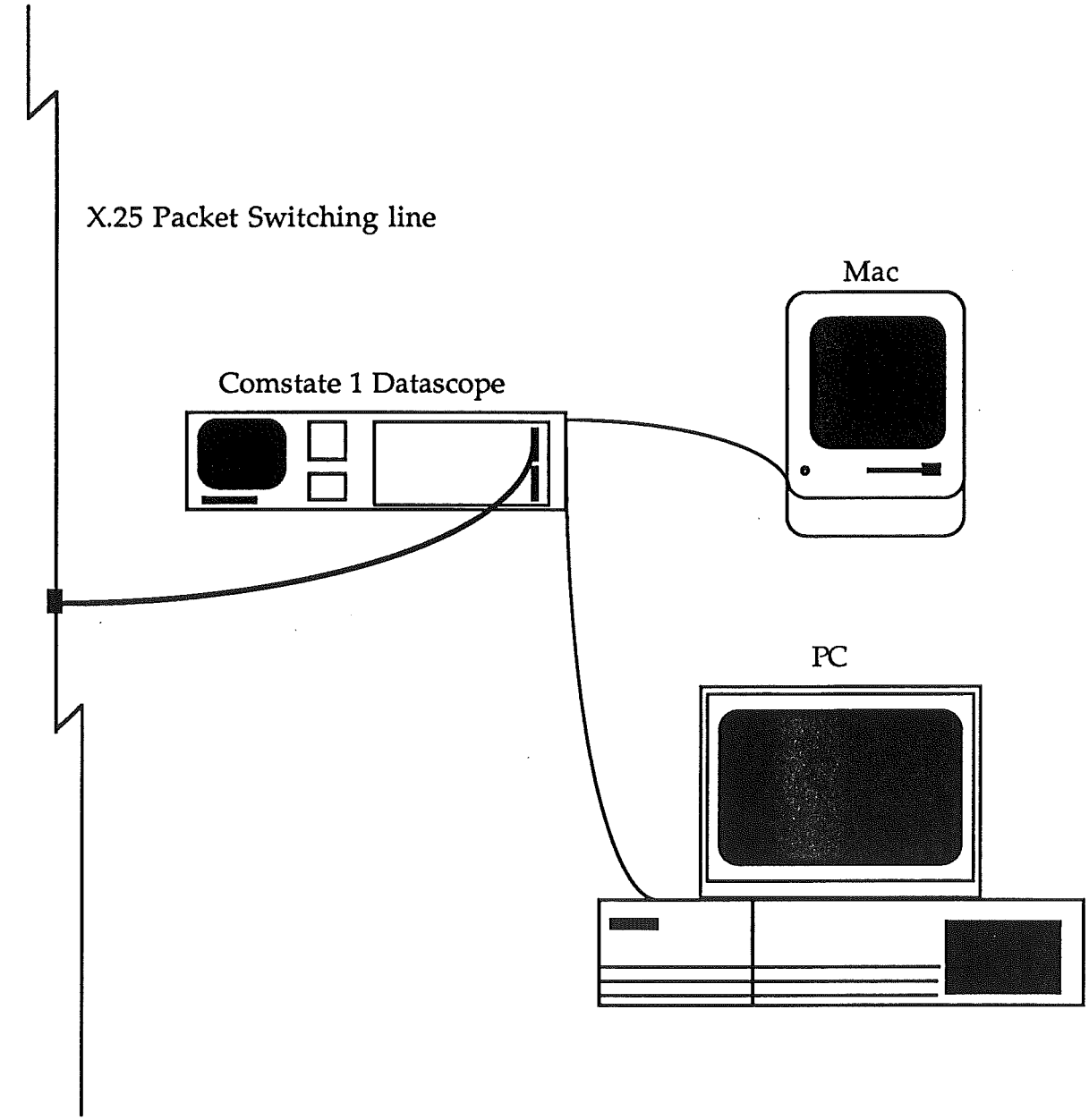


Figure 2 : Cable to connect Comstate 1 to Macintosh for print transfer

Figure 3 : Interconnection of the Hardware Used



2.3 The Existing Software

The software used in this project is as follows:

Lightspeed Pascal Version 1.11 : Lightspeed Pascal was chosen as the initial development environment because it provided as its core a standard Pascal implementation. It also has an excellent debugging environment (stops, steps and observe window) and when combined with the MacsBug general purpose debugger provided a reasonable development environment. The times when a trial run crashed the Mac, without giving some indication as to what the cause was, were limited. It was found that to bring the Macintosh environment to a stage were it can not recover from errors within a program and forces a re-boot is very common during development of an application. This is especially true once toolbox routines are being used. Due to the lack of examples in Inside Macintosh [3], a lot of knowledge about the toolbox routines has to gained from experimentation with code.

Extenders : Extenders provide an invaluable tool to the first time Macintosh programmer. It provides access to window and menu routines at a level well above the toolbox routines on which it is built. Due to the well laid out and graduated examples provided in the Extenders manual the developer is gradually eased into programming in the Macintosh environment.

Turbo Pascal Version 5 : The porting of the basic Pascal code from Lightspeed to Turbo proved simple enough. The only major change was in the method of opening the files required. In the Macintosh version this was done using the standard dialog boxes, so required changing for the Turbo version. The development of a pull-down-menu interface in the DOS environment is nowhere near as trivial as in the Macintosh environment. As such this was avoided and a set of graphical menu routines were developed. The main problem encountered in developing graphics routines for PC's and compatibles is the wide range of different graphics cards around.

MacScrewn Version 3.1 : This was used only as a means of capturing the print file obtained from the Comstate 1 Datascope, on the Macintosh. By using the record facility a copy of the data coming in the modem port can be obtained in a plain text file. MacScrewn is a terminal emulator for the Macintosh, developed at the University of Canterbury by J Collier and B J Mckenzie.

MicroTex Version 2.52 : This was used to capture the print file obtained from the Comstate 1 Datascope, on the PC. The capture facility of MicroTex was used to obtain a data file containing a transcript of the data coming in Coms port 1. MicroTex is a general communications package for the PC, developed in Christchurch by Technology Solutions Ltd.

2.4 The Development of the Software

2.4.1 Data File Format

Before any development of software could commence, a file was required in a format that was possible to interpret and validate. As the Comstate 1 does not support on-line transmission of the traffic which it is analysing, the only data available is that stored in the data buffer.

There were several options available for obtaining the data captured by the Comstate 1. It has the ability to down load the contents of the data buffer to a storage device (e.g PC) for latter up-loading back into its data buffer for re-analysis. Along with the actual data captured data from the line is extra data required by the Comstate 1 if it is to be able to re-analyse the data. This extra data is devoted to the attributes of the data characters. Due to this extra data, and a format (hexidecimal encoded ASCII) which was fairly undecipherable when desk checking the software, this format was abandoned.

The other means of obtaining the data from the data buffer is in the form of a print file. This means the Comstate 1 carries out the following file formatting:

(i) : File has three line header, 2 blank and 1 description.
(ii) : Dual format with a blank line between each pair of lines.
(iii) : Traffic from DTE to DCE first line of the pair.
(iv) : Traffic from DCE to DTE second line of the pair.
(v) : File ends with a blank line.

```
        ь
(i)     ь
        ASCII/8/NONE/SYNC/16 16        LN=0000
        ..................................................
(ii)    ex bs dl sh nu ( 1 ) sp sp sp sp sp A 1 e

(iii)     ..................................................
(iv)      x a n d e r sp t h e sp G r e a t
```

Figure 4 : Example of a print file transfered from the Comstate 1

Because all the characters in the file have to be printable the Comstate 1 substitutes unprintable characters with mnemonics (shown in table 3) and "sp" for space.

All these features are relied upon by the analysis software written. The three line header is checked to see that it matches the format expected, and if it does the file is taken to be of standard format. This means that if the file has been altered, e.g. using a text editor to concatenate two files, then care must be taken to maintain the expected format.

MNEMONIC	HEX VALUE	MNEMONIC	HEX VALUE
NU	00	DL	10
SH	01	D1	11
SX	02	D2	12
EX	03	D3	13
ET	04	D4	14
EQ	05	NK	15
AK	06	SY	16
BL	07	EB	17
BS	08	CN	18
HT	09	EM	19
LF	0A	SB	1A
VT	0B	EC	1B
FF	0C	FS	1C
CR	0D	GS	1D
SO	0E	RS	1E
SI	0F	US	1F

Table 3 Mnemonic Substitutions

The mnemonics come from [1] and are to equivalent to CCITT Alphabet No. 5

2.4.2 Obtaining a file from the Comstate 1

Once the method for obtaining the data file from the Comstate 1 was selected (i.e. print file) a method of capturing this file on the Macintosh and the PC was required. Software capable of this was eventually found in the form of MacScrewn, for the Macintosh and MicroTex for the PC. In both pieces of software the facility being used is for the recording of a session for re-analysis latter (e.g. login session or reading electronic mail).

When transfering a file, the speed of the Comstate printing must be matched to the speed MicroTex or MacScrewn is expecting. This is done by altering the "SPEED" selection in menu 5 of the Comstate 1 (see Fig 5).

```
      **V.24 INTERFACE CNTL/PRINTER**
      PRINTER CONTROL
→     SPEED: 9600
      NEW LINE: crlf
→     FOLLOWED BY 0  PADS
      CHAR PER LINE: 120
      V.24 INTERFACE CONTROL
      STATIC LEADS: DTR DSR
                    X  X
      RTS: ON
      CTS: ON
      RLSD: ON
      XMIT DELAY: 0  mSECS
```

Figure 5 : Comstate 1 Printer Menu.

2.4.2.1 File Capture on the Macintosh.

1 : Using the cable, as shown in Fig 2 , connect the printer port on the back of the Comstate 1 to the modem port of the Macintosh.

2 : Match the baud rate from the baud menu of MacScrewn to the baud rate in the printer set-up menu of the Comstate 1 (see Fig 5). The rest of the selections for the printer can be left the same.

3 : Display the data buffer (control F) of the Comstate 1.

4 : Select record and supply a file name in MacScrewn.

5 : Select Control-Print on the Comstate 1. This prints the entire contents of the data buffer.

6 : After the printing has finished stop the recording in MacScrewn.

2.4.2.2 File Capture on the PC.

1 : Using the cable supplied with the Comstate 1 connect the printer port on the back of the Comstate 1 to coms port 1 of the PC.

2 : Run MicroTex on the PC.

3 : Select "ASCII mode connect" (option 2) from the MicroTex menu. This takes you into a dialing directory in which a selection with even parity, 7 bits per character, 1 stop bit and reading from coms port 1 needs to be made.

4 : On the Comstate 1 the printer menu has to be set to match the speed of the MicroTex selection and the number of pad characters after a carriage return set to zero (see Fig 5).

5 : After establishing a connection with these parameters, ALT-C turns capture on.

6 : Display the data buffer (control-F) and print its entire contents (control-print).

7 : After the print has finished stop the capture in MicroTex (ALT-C).

When transfering a file from the Comstate to the PC the default setting of 5 pad characters following CR has to be changed to 0. This is because, unlike MacScrewn, MicroTex does not ignore the padding characters and they are written to the file. If the number of pads is not set to zero the file will not be recognised to be of the standard format expected.

NOTE : The copy of MicroTex provided has had all but one of the modem and printer drivers removed as they are not necessary for capturing files from the coms port.

2.4.3 Using the Macintosh Version of the Software Analyser

After Starting the application the following is the sequence commands to analyse a file:

1 : Select analyze from the RUN menu.

2 : Open the file you wish to analyse (in the std open dialog box).

3 : Accept or change the name of the file you want the text results written to.

4 : Select the bit rate of the line from which the sample came.

5 : Wait .

(for a more detailed explanation of the commands see below)

The numeric results of the analysis are displayed in the window 'Analyzer text wind' on the conclusion of the analysis. These results have also been written to the file of the selected name. A graphical representation of the results is also available by selecting the graph windows on the 'Window' menu. The graphs are not saved automatically like the numeric results are, although any graph may be saved by selecting 'save' or 'save as' while the required graph is the current window. A MacPaint format pict file will be saved with the name supplied by the user.

The text window can be used to open any text file. There are no prompts for saving changes when an action will destroy an unsaved buffer because all the analyser's text results are already saved.

Selecting a standard format file

If a file which does not have the expected header is selected in the first dialog box of the run sequence then the following error message is displayed.

Not A Standard Format Text File

The first three lines of the file obtained from the Comstate 1 should appear as follows.

```
b
b
ASCII/8/NONE/SYNC/16 16        LN=0000
```

To check this open the file in the text window and inspect it visually.

Selecting the bit rate of the line

The dialog box shown in Fig 6 is the fourth selection in the sequence to analyse a file. It is used to obtain the transmission rate of the line from which the sample came. This is used for the purposes of scaling the axis of the inter packet time graph to be in seconds. If the speed of the line is not known, or an indication of the number of character slots between data packets is required, the last selection of

"None / Do Not Know" can be chosen. If this option is chosen the scale of the horizontal axis of the inter packet time graph, is in character slots, instead of seconds.

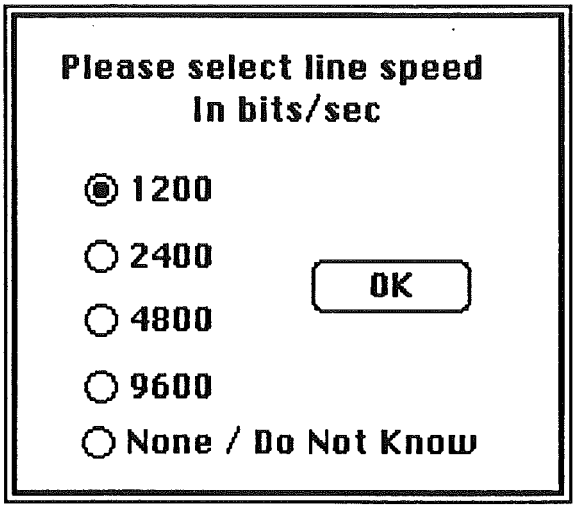


Figure 6 Select bit rate Dialog box

Analyse Next in Series

This selection is for analysing a series of samples taken from a line. Use the "analyse" selection for the first sample and the "Analyse next in Series" for subsequent samples. After all the samples have been analysed, selecting "Summary of series" displays a dialog box, the contents of which are explained in the Results section. Selecting the "Print Screen" button in the dialog box does a screen dump. Selecting "Analyse" rather than "Analyse next in Series" at any stage resets the statistics for the series to contain just the sample being analysed.

On-line help

On-line help is available by selecting the help button in the 'About Analyzer' dialog. This provides a help system similar to Word, with scrolling lists of help topics and text. If the file "Analyser help" is not in the same folder as the application when it is started up, the dialog box below is displayed (Fig 7). It is also displayed on any subsequent attempts to enter the help system. To rectify this, quit and find the help file, shift it into the same folder as the application and restart.

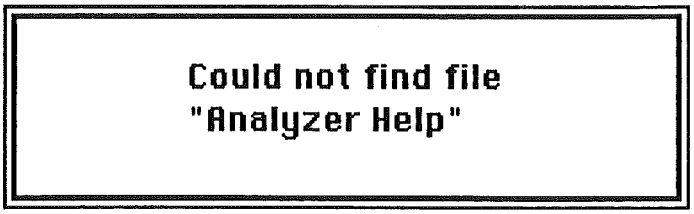


Figure 7 Error Dialog

The help is presented as scrollable lists in dialog boxes . To obtain help on a topic, select it and click "OK". Another dialog with help the selected topic is displayed over the top of the topic dialog box.

2.4.4 Using the PC version of the Software Analyzer

If hard copies of screens are required then type "graphics" before starting up the program. Screen dumps can then be sent to the printer by typing "shift-Print screen". After starting the application the following is the sequence commands to analyse a file :

1 : Select Analyze from the main menu

2 : Supply the full path name of the file to be analyzed, either from the root directory or from the current directory (shown as prompt).

3 : Supply a file name for the numeric results to be written to. A file name of 5 characters or less will make the names of the files produced by the analysis more understandable.

4 : Select the bit rate of the line the sample came from.

5 : Wait.

Carriage return is used as the method of entering a command after a selection has been made either on a menu or after text input .

The numeric results are displayed, like a Unix "more", 20 lines at a line, pausing for a carriage return from the user at the end of each section.

Selecting the bit rate of the line

This is used to obtain the transmission rate of the line the sample came from. This rate is used for the purposes of scaling the axis of the inter packet time graph to be in seconds. If the speed of the line is not known, or an indication of the number of character slots between data packets is required, the last selection of "None / Do Not Know" can be chosen. If this option is chosen the scale of the horizontal axis of the inter packet time graph is in character slots instead of seconds.

The "*Is this sample another sample in the same series*" question

This question is asked after the second, and any subsequent, analysis. This question is to allow for analysis of a series of samples taken from the same line. An answer of "yes" to this question adds the results of this analysis to the statistics being gathered. An answer of "no" to this question clears the statistics being gathered and then adds this analysis as the first of a new series. After all the samples have been analysed, selecting "Display Replication Res" displays a result screen, the contents of which are explained in Section 3.3.

DOS shell

A DOS shell is provided within the program for changing directory, obtaining directory listing etc. The prompt displayed, when a file name is required as input, is the current working directory. If the file for analysis is in a different directory or a sub-directory of the current one then a path from the current directory or starting from "\" must be given. The alternative is to use the DOS shell to change directories.

On Line Help

When "Help" is selected from the main menu changes the prompt at the bottom of the menu and subsequent selections bring up help on that subject. Selecting "Quit" from the main menu takes the program back to ordinary menus. If the file “ana2.hp” is not in the same directory as the progam then the help system will display an error message each time an attempt is made to use it. To rectify this quit and move the help file.

3 Results

The Analyzer produces three files, 6 graphs (see Fig 9 - 14) and a Summary dialog box (see Fig 15).

3.1 The Files Produced by the Software Analyser

The main file, which is written to the output file name provided, contains the text results displayed as the Analyser finishes analysing and returns to interactive mode (see Fig 9). This consists of a repeated set of statistics. The first set is from the traffic sent from the DTE to the DCE, the second is from the traffic sent from the DCE to the DTE.

Each of these sets consists of some line tallies followed by tallies for each active logical channel.

The line tallies are as follows :

Send Fill Characters : This is the number of character slots which are empty and are as such filled with idle characters. In the print file each idle slot is represented by three '.'. So '. . . . . .' , being six '.'s, would count two towards the total of the fill characters.

Send Control Count : This is a count of all the control characters in the file. It includes all layer two headers, trailers and control frames as well as layer three headers and control packets. It is a count of all the characters excluding user data and fill characters.

Send User Data Count : This is a count of the user data characters (e.g. data passed down to layer 3 for transfer) that has been sent in the sample. This is the count of the amount of useful data transferred in this time.

Send L2 Data Packet Count : This is a count of the number of layer two data frames which contain data which is destined for layer three. This data for layer three could be user data or just layer three control information e.g. a layer three RR.

Send L2 RR Packet Count : This is a count of the number of layer two Receive Ready frames. These frames contain only control data and are for updating the counters of received frames at the end they are being sent to.

Send L2 RNR Packet Count : This is a count of the number of layer two Receive Not Ready frames. These frames contain only control data and are for indicating an inability to accept additional information frames.

Send L2 REJ Packet Count : This is a count of the number of layer two Reject frames. These frames contain only control data and are for indicating the reception of an invalid packet.

Send L2 Other Packet Count : This is a count of the number of layer two frames of types not included in the counts above. These are frames of type
DISC (disconnection)
UA (Unnumbered Acknowledgement)
FRMR (Frame Reject)
SABM (Connection and Reset LAPB)
DM (Indication of Disconnected mode)

Percentage of Idle Send Line Time : This is the percentage of time the line is being filled with Idle characters, e.g. percentage of line capacity unused.

The logical channel tallies are as follows :

Logical Channel Number : Which logical channel the tallies are for.

Logical Channel Data Character Count : A count of how many user data characters were transmitted down this logical channel in the sample. A proportion of the Send User Data Count (see above) for the line.

Logical Channel Layer 3 Data Packet Count : A count of the number of data packets, for this logical channel, in the sample.

Logical Channel Layer 3 RR Packet Count : A count of the number of Receive Ready packets for this logical channel. These packets contain only control data and are used to update the counters of received packets at the end to which they are being sent.

Logical Channel Layer 3 RNR Packet Count : A count of the number of Receive Not Ready packets for this logical channel. These packets contain only control data and are for indicating an inability to accept additional data packets. e.g. flow control.

Logical channel layer 3 other packet count : A count of the number of packets for this logical channel of types which are not included in the counts above. These include
: Reset, Clear, Call, Restart and Incoming Call packets.

The sum of the four packet counts above summed over all the logical channels should be the value of the Send L2 Data Packet Count.

The other two files produced by the software analyser are given the name of the main file chosen, with a suffix. The two suffixes used are "dte.sp" and "dce.sp". These two files contain the numeric results used to draw the histograms. The format of the file is the same as expected by most spreadsheet type packages (like Cricket Graph and Excel). The two columns of numbers are separated by a tab. The first column is the inter-data packet gap and the second column is the data packet length. (see Fig 8). The reason for producing these files is to allow the user to use the data to produce more complex graphs, if required. It also allows the concatenation of the statistics from several samples to produce a summary graph.

Interpacket	Packet Length
5	128
66	130
55	130
28	130

Figure 8 The format of the spreadsheet files.

3.2 The Graphs produced by the Software Analyser

Six graphs are produced, three for each of the two directions. The data for the graphs is from all logical channels.

3.2.1 Histograms of Inter Data packet times

(see Figs 10 and 11)

This is a histogram of the time between data packets. The time between data packets is calculated by counting the number of character slots between the end of the user data in the previous packet to start of the user data in the next data packet. This total is then divided by the character rate (from the value supplied by the user) , the resulting value being an inter packet time in seconds. If the None / Don't Know option is selected on the Bits/sec menu then on the graph the time between data packets is in characters not seconds e.g. the characters/sec is set to 1. The time is taken between consecutive data packets regardless of which logical channel the data packet is for. If the graph appears and it is blank it means there were no data packets sent in the sample in that direction.

3.2.2 Histograms of Data packet lengths

(see Figs 12 and 13)

This is a histogram of the length of the user data fields of data packets in characters. As with the inter-data packet histogram, the data packets for all logical channels are lumped together. The main reason for doing this is the small size of the samples, making a graph for each logical channel very sparse. If the graph appears and it is blank it means there were no data packets sent in the sample in that direction.

3.2.3 Pie Graphs

(see Figs 14 and 15)

The pie graphs have three main types of slices. These represent the three kinds of characters, Data, Fill (idle), and Control. The Control slice represents the proportion of characters that are transmitted in frame and packet headers and in control frames and packets. These represent all the characters transmitted excluding user data and fill characters. The Fill slice represents the proportion of character slots that are empty (between frames) and are so filled with idle characters. This represents the amount of unused line capacity. Each data slices represents the proportion of user data characters that are transmitted in the sample for that logical

channel. If the statistics for a logical channel appear in the text results and it does not show up on the pie graph it means there was no user data transmitted for that logical channel in the direction specified.

3.3 The Summary of a Series of Samples

(see Fig 16)

The reason for providing this facility is to help in overcoming the small size of the samples available. By averaging the percentage of each type of character over several samples a reasonable estimate of the true mean load on the link can be obtained. The weight placed on each sample is the same, regardless of the size of the sample. The mean and standard deviation of the percentage fill, after the analysis of several samples from the same link, are the most meaningful results produced by the software analyser. These figures give a very good indication as to the percentage of link capacity unused in the time period the samples were taken.

By randomly sampling the link and then analysing the samples as a series, an average load during the day can be obtained. There is probably more information to be obtained by analysing the link during the peak half hour of usage, or times of worst response times of the communications network.

3.4 Example Results : File and Graphs

Figures 9 to 15

These are the file and graphs produced from analysing a sample containing the following:

(i) : A file transfer from DTE to DCE on logical channel 4.
(ii) : A file transfer from DTE to DCE on logical channel 1.
(iii) : A file transfer from DCE to DTE on logical channel 1.

Because the sample contained file transfers the average packet size is close to the maximum packet size of 128 bytes. This is because there is always enough data to fill the packets until the last part packet to finish the file. This is also the reason for the short inter data packet times on average. There is always data ready to transmit and the rate is only limited by the flow control of the link.

```
Results of analysing Colin Daniell:Bigger One
DTE Results
send fill count                                 =   547
send control count                              =   419
send user data count                            =  4200
send L2 data packet count                       =    54
send L2 rr packet count.                        =     9
send L2 rnr packet count                        =     0
send L2 rej packet count                        =     0
send L2 other packet count                      =     0
Percentage of idle send line time               = 10.59%

Logical channel number                          =     1
Logical channel data character count            =  2104
Logical channel layer 3 data packet count       =    18
Logical channel layer 3 rr packet count         =    14
Logical channel layer 3 rnr packet count        =     0
Logical channel layer 3 other packet count      =     0

Logical channel number                          =     4
Logical channel data character count            =  2096
Logical channel layer 3 data packet count       =    18
Logical channel layer 3 rr packet count         =     0
Logical channel layer 3 rnr packet count        =     0
Logical channel layer 3 other packet count      =     4


DCE Results
send fill count                                 =  2558
send control count                              =   501
send user data count                            =  1988
send L2 data packet count                       =    55
send L2 rr packet count                         =    28
send L2 rnr packet count                        =     0
send L2 rej packet count                        =     0
send L2 other packet count                      =     0
Percentage of idle send line time               = 50.68%

Logical channel number                          =     1
Logical channel data character count            =  1988
Logical channel layer 3 data packet count       =    17
Logical channel layer 3 rr packet count         =    16
Logical channel layer 3 rnr packet count        =     0
Logical channel layer 3 other packet count      =     0

Logical channel number                          =     4
Logical channel data character count            =     0
Logical channel layer 3 data packet count       =     0
Logical channel layer 3 rr packet count         =    18
Logical channel layer 3 rnr packet count        =     0
Logical channel layer 3 other packet count      =     4
```

Figure 9 Example of main results file

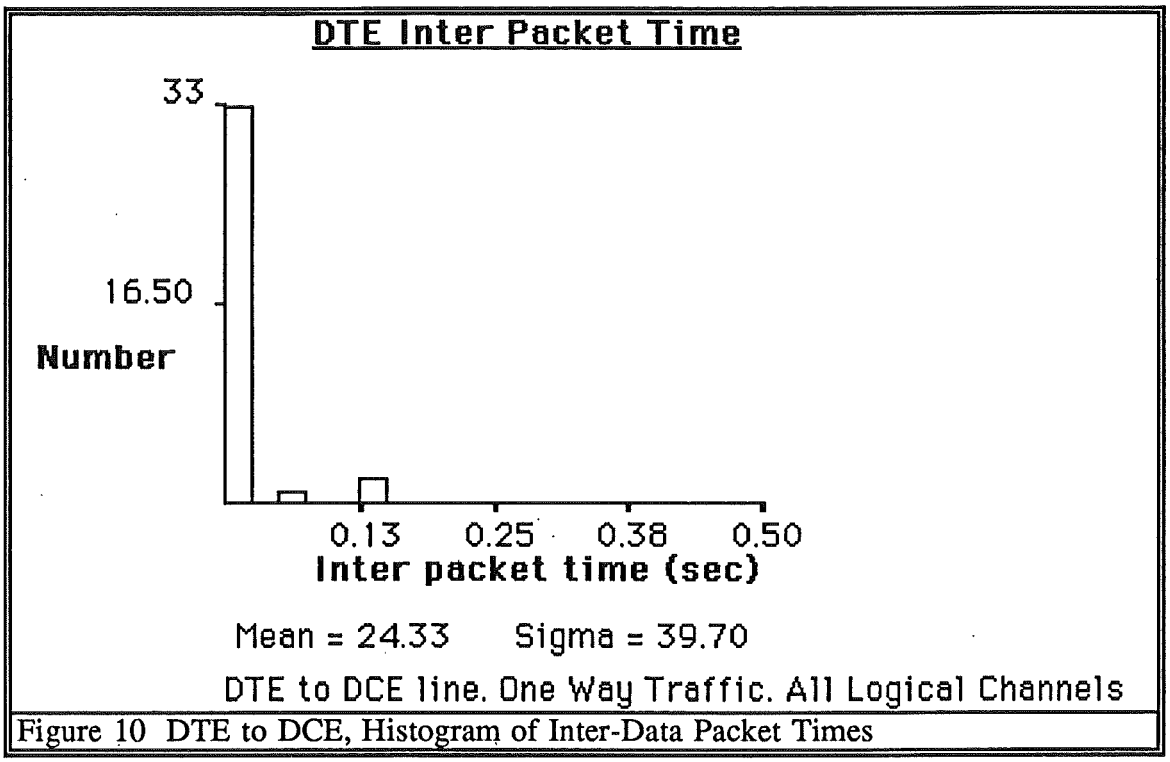


Figure 10 DTE to DCE, Histogram of Inter-Data Packet Times

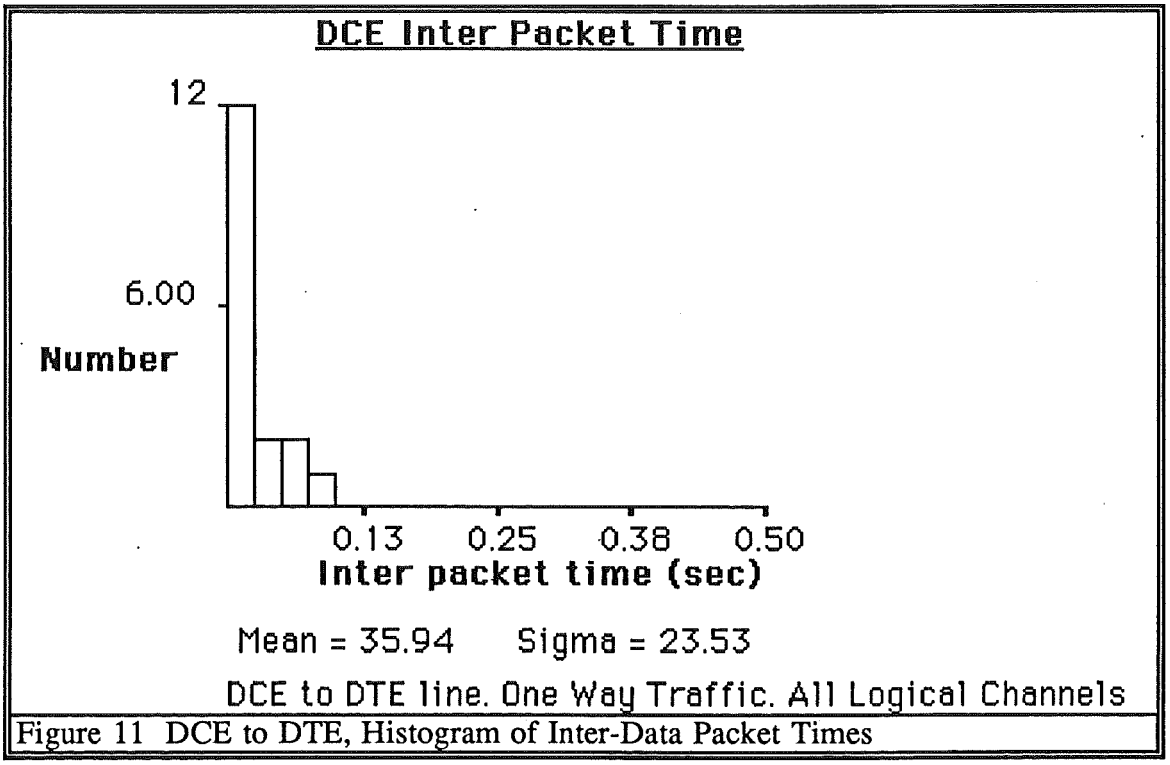


Figure 11 DCE to DTE, Histogram of Inter-Data Packet Times

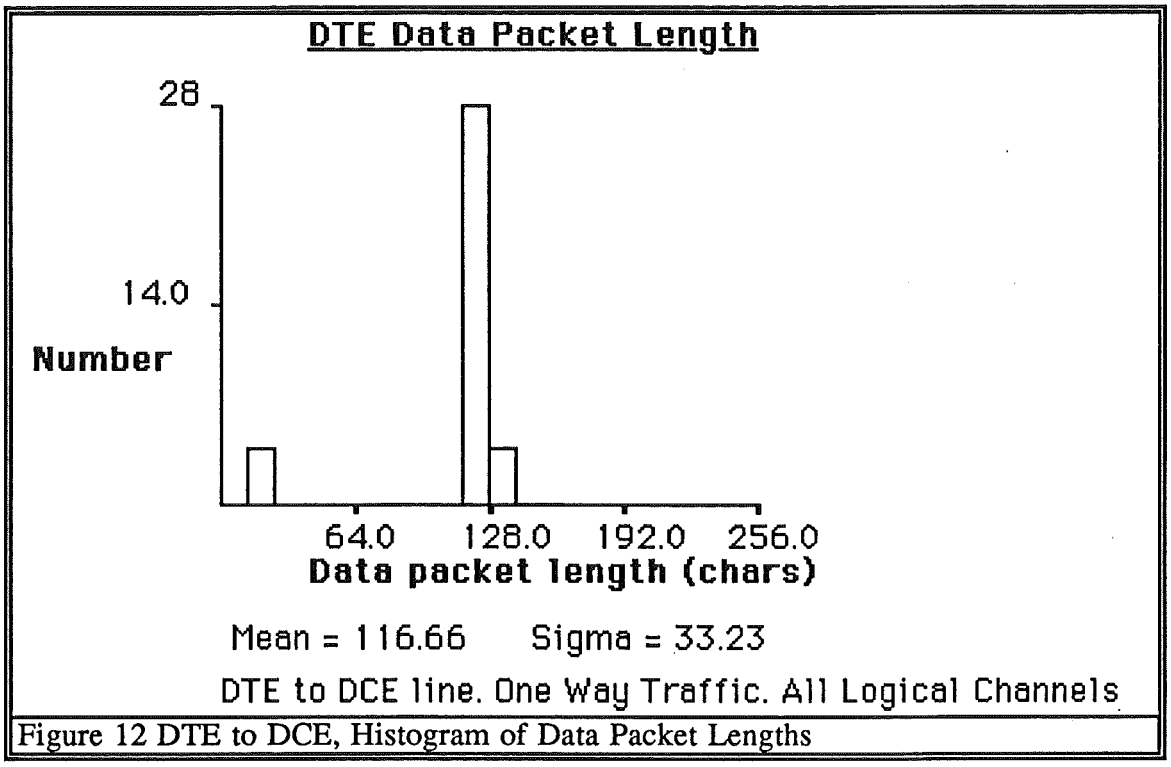


Figure 12 DTE to DCE, Histogram of Data Packet Lengths

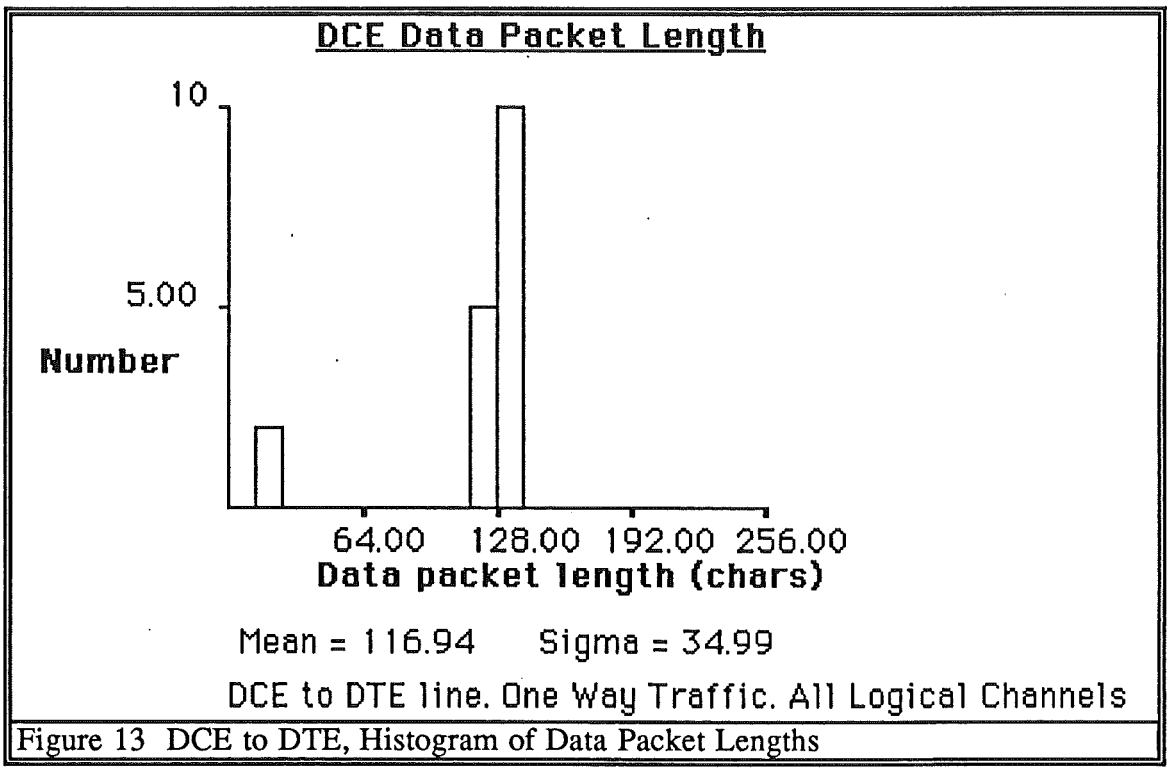


Figure 13 DCE to DTE, Histogram of Data Packet Lengths

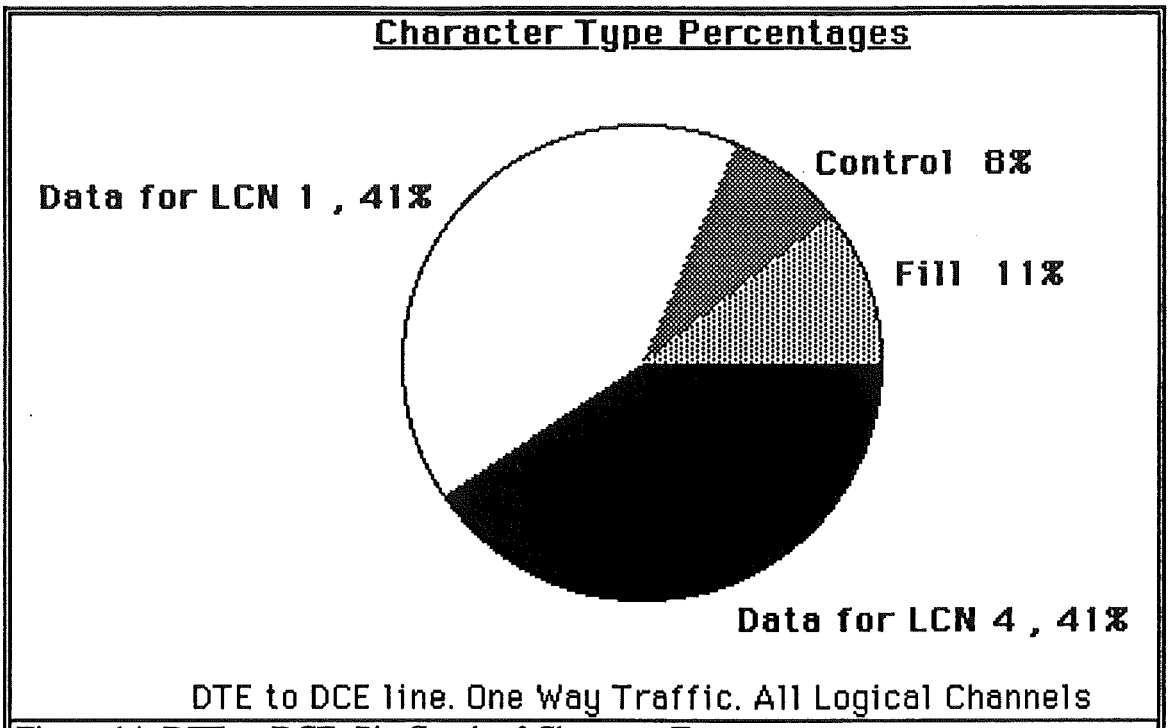


Figure 14 DTE to DCE, Pie Graph of Character Types

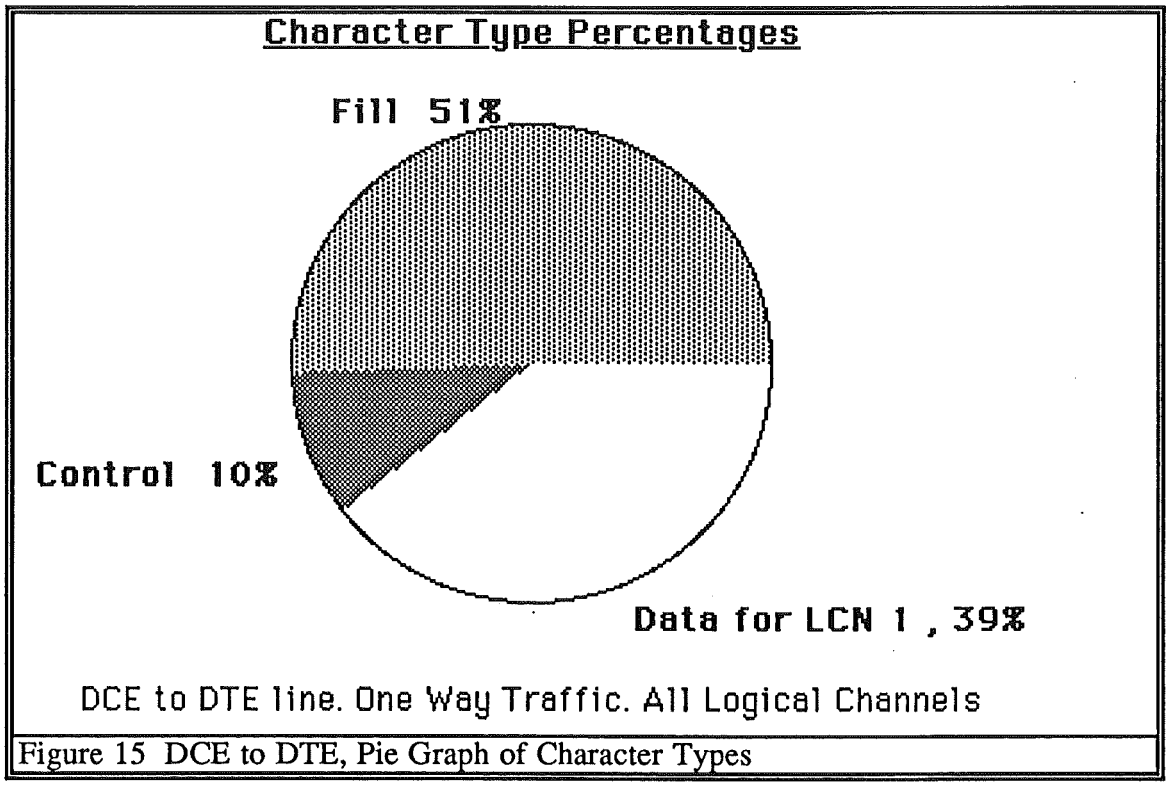


Figure 15 DCE to DTE, Pie Graph of Character Types

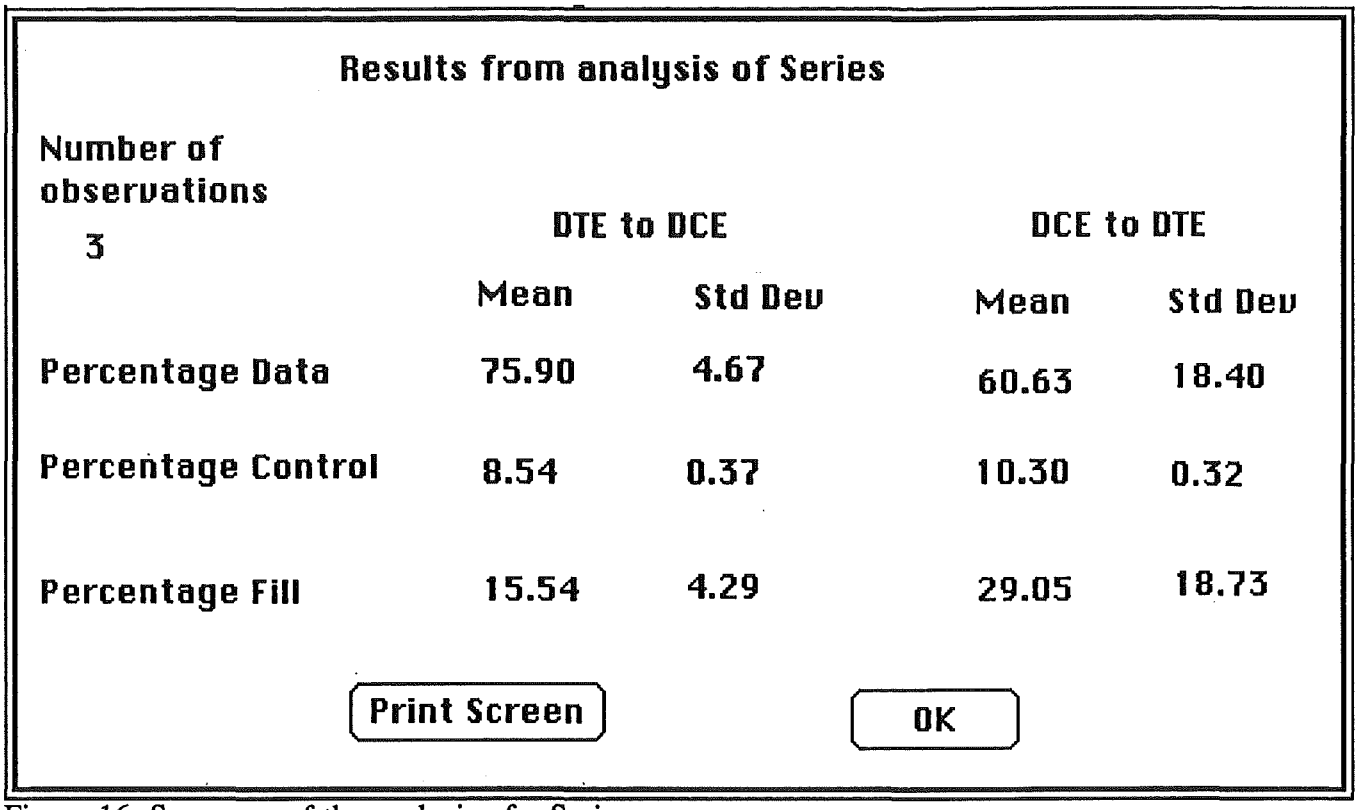


Figure 16 Summary of the analysis of a Series

4 Conclusion

The aim of developing a tool to aid with the statistical analysis of the data flow on communications lines operating with the X.25 packet switching protocol was accomplished. With the use of micro-computer based software this package extends the statistical analysis capabilities of Comstate 1. The hardware, existing software and the software developed combine to provide a user friendly package on the PC and the Macintosh. The graphical portrayal of the results make the interpretation of them quick and easy.

The major problem with this package as a whole is the limited size of the samples it is possible to obtain from the Comstate 1. There appears no way around this problem with the hardware available. The analysis of several small samples and the use of mean values produced goes some of the way towards overcoming the problems of the small size of the samples being analysed.

Other forms of statistical analysis, like the calculation of the statistical distribution the observations, were not achieved in this project. The two major reasons for this were the limited time for the development of the software and the limited size of the samples it is possible to capture. Development of the software could not start until the method of capturing the print file from the Comstate 1 on the PC and Macintosh was finalised. This was held up by the search for software capable of performing the necessary file capture. Due of the hardware imposed limitations on the size of the samples, the number of observations on which to preform these calculations would be restricted. If it had been possible to obtain larger samples through the use of a different datascope other forms of statistical analysis would have been added to the package developed.

Obtaining 'real' traffic samples to test the software analyser on proved difficult. The main reasons for this is the confidential nature of a lot of the traffic on packet switching connections and the need to interrupt the link to connect the Comstate 1 in and again to disconnect it. For these reasons the software was tested with artificial samples generated in the laboratory by connecting two PC's together and using X.25 protocol to send files back and forwards. Even though the traffic samples produced were not totally 'real' they provided a reasonable test for the software because they covered most most of the variations expected.

The software analysis package developed in this project was checked against hand counts of these samples and found to be accurate.

Alternative means of Statistical Analysis of an X.25 Link.

The latest models of X.25 protocol analysers (datascopes) available far exceed the capabilities of the Comstate 1 [4]. Most are programmable in one or two languages (C, Basic, FOURTH, plain language) giving the analysts easy access to combinations of their facilities in repeatable tests. Due to internal disc storage most of the new datascopes are capable of capturing large samples of traffic (up to 96 hours of full duplex 9.6 kbit/s line at 100 percent utilization). They are also capable of preforming statistical analysis of these captured samples including:

* Line utilization.
* Response time
* Peak data-traffic periods.
* Packet statistics : shortest, longest and average packet length.
* HDLC frame counts.

These statistics are usually provided in a graphical interface with a range of bar charts and graphs. Although the capabilities of these protocol analysers exceed that of the package developed in this project, their performance is reflected in their cost. At around $50,000 they are beyond the budget of all but network providers.

This package is presented as an alternative to the high capital investment required for a modern X.25 protocol analyser. From the descriptions of the facilities these hardware based X.25 protocol analysers provide the software package developed produces similar results, the only major difference being the size of sample it is possible to work with.

5 References

[1] *Technical Manual for Interview® Comstate™ 1*, Atlantic Research Corporation, Issue 4, September 1985 .

[2] *Packet Switching Data Transmission Service, Packet Mode, Customer Interface Specification*, New Zealand Post Office.

[3] Apple Computer, *Inside Macintosh*, Vols 1, 2, 3. 4 and 5, Addison Wesley 1987.

[4] Hird E.V. Making the most of your X.25 protocol analyzer, *Data Communications*, Vol.17, No.2, February 1988, pp 165 - 176.

[5] Kreutzer W. *System Simulation, Programming Styles and Languages*, Addison Wesley, 1986, pp 254 - 262.

6 Appendix 1 Code Listings

Some of data structures for the storing of statistics and the code for the manipulation of these structures came from [5] (e.g. Atallytype, Ahistogramtype).

Summary of Major Data Structures :

Ahistogramtype

title	The title to be displayed at the top of the graph.
hor_title	The title of the horizontal axis.
NoOfClasses	The number of classes to divide into.
classwidth	The width of each of the divisions
low	The value at which observations under-flow.
high	The value at which observations overflow.
tally	A record of type Atallytype
counters	An array of counts for # of observations in each class.
overflow	The number of observations which under-flow.
underflow	The number of observations which overflow.

This structure is used to store all the information necessary to plot a histogram. Changing the number of class alters the granularity of the resulting graph.

Atallytype

noOfObs	The number of observations recorded.
integral	The value of sum(X).
sumofsq	The value of sum(X^2).
minobs	The minimum observation recorded.
maxobs	The maximum observation recorded.

A record for keeping track of information for the calculation of the mean and variance of the observations that are added to it.

Statstruct

phist	histogram of data packet lengths
ihist	histogram of inter data packet times
infoheader	value of first byte of header
frametype	type of frame currently in
fillcount	count of fill characters
logicalchannel	pointer to linked list of LC's
this_lcn	ptr to the LCN node of the current packet
L2packets	count of layer 2 frame types
controlcount	count of control characters
datacount	total number of data characters
oschars	count of outstanding characters (chars since last space)
headcount	count of how many characters seen in current header
presentlyin	type of data reading at the present
interpacket	count of inter-data-packet characters
thispacket	count of characters in this packet
thisfill	count of characters in this fill section

graph	file of statistics for spreadsheet

This structure is used to store all the information about the analysis of the sample. One of these is created for each direction (DTE to DCE, DCE to DTE). This structure contains all the counters for maintaining the statistics as well as the state information. The state information is required because of the switching from one direction to the other at the end of each line.

Lcnstruct

lcn_number	logical channel number
L3packets	count of layer 3 packets for this lcn
charactercount	count of characters for this lcn
next	ptr to next lcn record

This record is used to store the information about one logical channels layer three data.

Packetype

data	count of layer's data packets.
rrpack	count of layer's RR packets.
rnrpack	count of layer's RNR packets.
rejpack	count of layer's REJ packets.
other	count of layer's packets of other types.

This record is used to store the information about one logical channels layer three packet counts.

Macintosh Code

The code for the Macintosh version of the Analyser was written in Lightspeed Pascal version 1.11. The Programmer's Extenders libraries were used as a bases from which to start in the development of the interface.

The first print out is the help file.

The code is divided up into five units :

MyTypes Unit This contains the types and constants that are used globally in the program.

Graph Unit This contains the procedures for manipulation and graphing of the of the statistics.

Help Unit This contains the procedures for initialising and displaying the help information using lists in dialog boxes.

PAnalyse Unit This contains the code which does the actual analysis of the file containing the sample.

Main This contains the main event loop for selection of menus and windows.

Analyse ;Histograms Inter Packet ;Histograms Packet size ;Pie Graphs; Analyse Next in Series; Summary of the Series;

1 Analyze :
To Analyze a traffic sample do the following.
Select Analyse from the RUN menu.
When the standard open dialog box appears select the file you what to analyze. The next dialog box is to obtain a file name to save the results to.
Two other files are also created (with extensions to this name) for use with speadsheets.
The BIT RATE asked for next is the bit rate of the line the sample came from. If "None" is selected the inter packet times are measured in characters not seconds.

2 Inter Packet Times Histogram :
This is a histogram of the time in seonds between packets containing user data. The scale of the horizontal axis changes with the bit rate selected. If the "None" option is selected then the horizontal axis is in characters between data packets and not in seconds.
It is the inter data packet time for data packets on any logical channel.

3 Packet Length Histogram :
This is a histogram of the length, in characters, of data packets. It covers data packets on any logical channel.

4 Pie Graphs :
The Three segments represent the following

FILL	This is the proportion of time the line is idle and so being filled with idles characters.
CONTROL	This is the proportion of time characters are being sent which are part of the protocol overhead eg not user data.
DATA	This is the proportion of time user data characters are being sent. There is one slice for the data on each logical channel.

5 Analyse Next in Series :
This is for analysing several samples from the same line to produce summary statistic over the series. This is useful if the samples captured are small. Chose 'Analyse' for the first sample in the series and 'Analyse Next in Series" for the rest of the examples.

6 Summary of the Series :
Selecting this displays a dialog box which contains the results of the analysis of the series so far. The mean is the percentage of that type of character found in the sample, averaged over all the samples analysed in the series so far. The standard deviation is an indication as to the variance of the values averaged.

```
{ This unit contains the global type and constant definitions and the global variables. }

unit mytypes;
interface
  uses
    XTTypeDefs;
  const
    runId = 20;                { ID of "RUN" menu }
    windId = 21;               { ID of "Window" menu }
    MaxHistClasses = 40;       { Max divisions in a histogram }

    dtefoot = 'DTE to DCE line. One Way Traffic. All Logical Channels';
    dcefoot = 'DCE to DTE line. One Way Traffic. All Logical Channels';
    pieheader = 'Character Type Percentages';

  type

    atallytype = record
        noOfObs : 0..maxint;
        integral, sumofsq : real;
        minobs, maxobs : real
      end;

    replications = record
        control, fill, data : atallytype;
      end;

    ahistogramtype = record
        title : string;
        hor_title : string;
        NoOfClasses : 0..MaxHistClasses;
        classwidth : real;
        low, high : integer;
        tally : atallytype;
        counters : array[0..MaxHistClasses] of 0..Maxint;
        overflow, underflow : 0..Maxint
      end;

    packetype = record
        data : integer;          (* count of layer's data's *)
        rrpack : integer;        (* count of layer's RR's  *)
        rnrpack : integer;       (* count of layer's RNR's  *)
        rejpack : integer;       (* count of layer's REJ's (not layer 3) *)
        other : integer;         (* count of layer's other frames *)
      end;

    nextptr = ^lcnstruct;
    lcnstruct = record
        lcn_number : integer;      (* logical channel number *)
        L3packets : packetype;     (* count of layer 3 packets for this lcn*)
        charactercount : integer;  (* count of characters for this lcn *)
        next : nextptr;            (* ptr to next lcn record *)
      end;

    statstruct = record
        phist : ahistogramtype;    (* histogram of data packet lengths *)
        ihist : ahistogramtype;    (* histogram of inter data packet times *)
        frametype : integer;       (* type of frame currently in *)
```

```
        fillcount : integer;          (* count of fill chars *)
        logicalchannel : nextptr;     (* pointer to linked list of LC's *)
        this_lcn : nextptr;           (* ptr to the LCN node of the current packet *)
        L2packets : packetype;        (* count of  layer 2 frame types  *)
        controlcount : integer;       (* count of control chars *)
        datacount : integer;          (* total number of data chars *)
        oschars : integer;            (* count of outstanding chars (since last space)*)
        headcount : integer;          (* count of how many chars seen in current header *)
        presentlyin : integer;        (* type of data reading at the present  *)
        interpacket : integer;        (* count of inter-data-packet chars *)
        thispacket : integer;         (* count of chars in this packet *)
        thisfill : real;              (* count of chars in this fill section *)
        graph : text;                 (* file of stattistics for spreedsheet *)
      end;

  var
    event : EventRecord;
    WhatHappened : EventStuff;
    dce, dte : statstruct;            { records for storing line statistics       }
                                      { Dce and dte are globals and not passed as  }
                                      { parameters to procedures that use or alter }
                                      { them as most procdures in the program either }
                                      { use the data in them or add data to them.  }
implementation
end.
```

```
{ This unit contains the procedure for drawing graphs on windows. It also contains  }
{ the procedures for maintaining histogram and tally statistics.                    }

unit graphs;

interface
  uses
    XTTypedefs, extender1, mytypes;

  function int_to_string (i : integer) : string;

  function real_to_string (num : real) : string;

  function TalliedSigmaQ (tally : atallytype) : real;

  procedure inittally (var tally : atallytype);

  procedure updatetally (var tally : atallytype;
                         obs : real);

  procedure inithistogram (var histo : ahistogramtype;
                           lowp, highp : integer;
                           classesp : integer;
                           name : string;
                           horzon : string);

  procedure updatehist (var histo : ahistogramtype;
                        obs : integer);

  procedure showhistogram (var wind : windowptr;
                           histo : ahistogramtype;
                           xcoord, ycoord, charsec : integer;
                           footer : string);

  procedure piegraph (fill, control : integer;
                      this : statstruct;
                      var wind : windowptr;
                      title : string;
                      footer : string);

implementation

{-------------------------int_to_string-----------------------}
  function int_to_string;
    var
      s : string;
  begin
    if i > 9 then
      begin
        s := concat(int_to_string(i div 10), chr((i mod 10) + ord('0')));
      end
    else
      s := chr(i + ord('0'));
    int_to_string := s;
  end;

{-------------------------real_to_string----------------------}
{ Real number to a string with two decimal places             }
```

```
  function real_to_string;
    var
      temp : string;
  begin
    temp := concat(int_to_string(trunc(num)), '.');
    num := (num - trunc(num)) * 10;
    temp := concat(temp, int_to_string(trunc(num)));
    num := (num - trunc(num)) * 10;
    real_to_string := concat(temp, int_to_string(trunc(num)));
  end;

{-------------------------TalliedSigmaQ--------------------- }
{ Calculate the standard deviation of the tally              }
  function TalliedSigmaQ;
  begin
    with tally do
      if (noofobs > 1) then
        TalliedSigmaQ := sqrt(abs((sumofsq - sqr(integral) / noofobs) / (noofobs - 1)))
      else
        TalliedSigmaQ := 0.0
  end;

{-----------------------inittally----------------------------}
{ Initialise a tally                                         }
  procedure inittally;
    const
      alargenumber = 32000;
  begin
    with tally do
      begin
        noofobs := 0;
        integral := 0.0;
        sumofsq := 0.0;
        minobs := alargenumber;
        maxobs := 0;
      end;
  end;

{-----------------------updatetally--------------------}
{Add an observation to the specified tally             }
  procedure updatetally;
  begin
    with tally do
      begin
        noofobs := noofobs + 1;
        integral := integral + obs;
        sumofsq := sumofsq + sqr(obs);
        if obs < minobs then
          minobs := obs;
        if obs > maxobs then
          maxobs := obs;
      end;
  end;

{-------------------------inithistogram---------------------}
{ Initialise a histogram                                    }
  procedure inithistogram;
    var
```

```
      index : 1..maxhistclasses;
  begin
    with histo do
      begin
        title := name;
        hor_title := horzon;
        noofclasses := classesp;
        low := lowp;
        high := highp;
        classwidth := (high - low) / noofclasses;
        inittally(tally);
        for index := 1 to noofclasses do
          counters[index] := 0;
        overflow := 0;
        underflow := 0;
      end;
  end;

{----------updatehist----------}
{Add an observation to the specified histogram }
  procedure updatehist;
    var
      class : 1..maxhistclasses;
  begin
    updatetally(histo.tally, obs);
    if obs < histo.low then
      histo.underflow := histo.underflow + 1
    else if obs > histo.high then
      histo.overflow := histo.overflow + 1
    else
      begin
        class := trunc((obs - histo.low) / histo.classwidth) + 1;
        if class > histo.noofclasses then
          class := histo.noofclasses;
        histo.counters[class] := histo.counters[class] + 1;
      end
  end;



{----------ShowHistogram---------- }
{ Displays a histogram at the required location with an X-axis width of 200 and a }
{ Y-axis height of 150. }
  procedure showhistogram;
    const
      height = 150;
      width = 200;
    var

      classlabel : real;
      percent : real;
      unitheight : real;
      unitwidth : integer;
      index, barlines : integer;
      maxcount : integer;
      R, bounds : rect;
      graphpic : pichandle;
```

```
{ Draw a bar of the histogram where num is number of units in bar and unitH and  }
{ unitW are the height and width of a unit. Draws it to the  right and above the  }
{ current location.}
    procedure drawbar (num : integer;
                       unitH : real;
                       unitW : integer);
    begin
      line(0, -trunc(num * unitH));
      line(unitW, 0);
      line(0, trunc(num * unitH));
    end;

{Adds the ticks and numbers to the horizontal axis}
    procedure addhorizon_axis (xcoord, ycoord, width, high, char_sec : integer);
      var
        i : integer;
        save_pen_state : penstate;
        num_sec : real;
    begin
      moveto(xcoord + width, ycoord);
      PenSize(2, 1);
      num_sec := high / char_sec;
      for i := 0 to 3 do
        begin
          GetPenState(save_pen_state);
          line(0, 3);
          move(-11, 14);
          drawstring(real_to_string(num_sec - (num_sec / 4) * i));
          SetPenState(save_pen_state);
          move(-(width div 4), 0);
        end;
      PenNormal;
    end;


  begin
    setrect(bounds, 0, 0, 500, 300);
    graphpic := openpicture(bounds);
    cliprect(bounds);

{draw axis}
    moveto(xcoord + width, ycoord);
    lineto(xcoord, ycoord);
    lineto(xcoord, ycoord - height);

{add titles and labels of axis}
    textface([bold, underline]);
    moveto(xcoord + width div 6, ycoord - (height + 22));
    drawstring(histo.title);
    textface(thePort^.txface - [underline]);
    moveto(xcoord + width div 6, ycoord + 30);
    drawstring(histo.hor_title);
    moveto(xcoord - 70, ycoord - height div 3);
    drawstring('Number');
    textface([]);
    moveto(xcoord, ycoord);

{add the bars}
```

```
    if histo.tally.noofobs > 0 then
      begin
        with histo do
          begin
            maxcount := counters[1];
            for index := 2 to noofclasses do
              if counters[index] > maxcount then
                maxcount := counters[index];
            unitheight := (height / maxcount);
            unitwidth := trunc(width / noofclasses);
            for index := 1 to noofclasses do
              drawbar(counters[index], unitheight, unitwidth)
          end;

{add the vertical and horizontal scales }
          moveto(xcoord, ycoord - height div 2);
          line(-3, 0);
          move(-(stringwidth(real_to_string(maxcount / 2)) + charwidth('H')), 0);
          drawstring(real_to_string(maxcount / 2));
          moveto(xcoord, ycoord - height);
          line(-3, 0);
          move(-(stringwidth(real_to_string(maxcount)) + charwidth(' ')), 0);
          drawstring(real_to_string(maxcount));
          addhorizon_axis(xcoord, ycoord, width, histo.high, charsec);
          moveto(xcoord, ycoord + 55);

{add mean and std dev at the bottom}
          drawstring(' Mean = ');
          drawstring(real_to_string(histo.tally.integral / histo.tally.noofobs));
          move(20, 0);
          drawstring(' Sigma = ');
          drawstring(real_to_string(TalliedSigmaQ(histo.tally)));
          moveto(xcoord, ycoord + 75);
          drawstring(footer);
        end;
      closepicture;
      setWpic(wind, graphpic);
    end;

{-------------------------------PieGraph-------------------------------------}
{ Displays a pie graph with its centre at the location specified by CentX and CentY }
{ and a radius of Rad. The method of working backwards around the circle starting   }
{ at +90 degrees is because the procedure was initially written on the PC and then  }
{ ported to the Macintosh.                                                          }
    procedure piegraph;
      const
        CentX = 250;
        CentY = 140;
        Rad = 90;
      var
        R, outside, bounds : rect;
        graphpic : pichandle;
        fillpercent, contpercent, datapercent : integer;
        total, start, X, Y, AdjX, AdjY : integer;
        angle_from_H : integer;
        contstring, datastring, fillstring : string;
        next_lcn : nextptr;
        shading : array[0..1] of pattern;
```

```
    count, data : integer;

{Calculate were to put text label }
    procedure GetTextCoords (AngleInDegrees : integer;
                             Radius : integer;
                             var X, Y : integer);
      var
        Radians : real;
    begin
      Radians := AngleInDegrees * Pi / 180;
      X := round(Cos(Radians) * Radius);
      Y := round(Sin(Radians) * Radius);
    end;

{ adjust the text position depending on whether on left or right of pie }
    procedure AdjustText (AngleInDegrees : integer;
                          wedgelabel : string;
                          var AdjX, AdjY : integer);
      var
        info : fontinfo;
    begin
      if (AngleInDegrees < 0) and (AngleInDegrees > -180) then
        AdjX := StringWidth(wedgelabel) + CharWidth('H')
      else
        AdjX := -CharWidth('H');
      if (AngleInDegrees < -90) and (AngleInDegrees > -270) then
        begin
          getfontinfo(info);
          AdjY := info.ascent + info.descent
        end
      else
        AdjY := 0;
    end;

{total the number of data characters sent on all the logical channels }
    function add_up_data (lcn : nextptr) : integer;
      var
        sum : integer;
    begin
      sum := 0;
      while lcn^.next <> nil do
        begin
          sum := sum + lcn^.charactercount;
          lcn := lcn^.next;
        end;
      add_up_data := sum;
    end;

  begin
    R := wind^.portrect;  {the rectangle bounding the window wptr}
    graphpic := openpicture(R);
    setrect(bounds, 0, 0, 500, 300);
    cliprect(bounds);
    textface([bold, underline]);
    moveto(centX - stringwidth(title) div 2, 20);
    drawstring(title);
    textface(thePort^.txface - [underline]);
    setrect(outside, 160, 50, 340, 230);
```

```
    shading[0] := white;
    shading[1] := black;

{calculate the percentages for the fill and control pie slices }
    total := fill + control + add_up_data(this.logicalchannel);
    fillpercent := round(fill / total * 100);
    contpercent := round(control / total * 100);
    fill := round(fill / total * 360);
    control := round(control / total * 360);

    contstring := concat(concat('Control ', int_to_string(contpercent)), '%');
    fillstring := concat(concat('Fill ', int_to_string(fillpercent)), '%');
    start := 90;
    angle_from_H := 0;

{draw in fill characters pie slice }
    if fill > 0 then
      begin
        penpat(ltgray);
        paintarc(outside, start, -fill);
        GetTextCoords(fill div 2, Rad, X, Y);
        AdjustText(start - fill div 2, fillstring, AdjX, AdjY);
        moveto(CentX + X - AdjX, CentY - Y + AdjY);
        drawstring(fillstring);
        start := start - fill;
        angle_from_H := fill;
      end;

{draw in control characters pie slice }
    if control > 0 then
      begin
        penpat(gray);
        paintarc(outside, start, -control);
        GetTextCoords(angle_from_H + control div 2, Rad, X, Y);
        AdjustText(start - control div 2, Contstring, AdjX, AdjY);
        moveto(CentX + X - AdjX, CentY - Y + AdjY);
        drawstring(Contstring);
        start := start - control;
        angle_from_H := angle_from_H + control;
      end;
    next_lcn := this.logicalchannel;
    count := -1;

{add a pie slice for the data characters, one for each logical channel}
    while (next_lcn^.next <> nil) do
      begin
        if next_lcn^.charactercount > 0 then
          begin
            count := (count + 1) mod 2;
            data := round(next_lcn^.charactercount / total * 360);
            datapercent := round(next_lcn^.charactercount / total * 100);
            datastring := concat(concat('Data for LCN ', concat(concat(int_to_string(next_lcn^.lcn_number), ' , '),
     int_to_string(datapercent))), '%');
            penpat(shading[count]);
            paintarc(outside, start, -data);
            GetTextCoords(angle_from_H + data div 2, Rad, X, Y);
            AdjustText(start - data div 2, Datastring, AdjX, AdjY);
            moveto(CentX + X - AdjX, CentY - Y + AdjY);
```

```
            drawstring(Datastring);
            start := start - data;
            angle_from_H := angle_from_H + data;
          end;
        next_lcn := next_lcn^.next;
      end;
    penNormal;
    frameoval(outside);
    textface(thePort^.txface - [bold]);
    moveto(centX - stringwidth(footer) div 2, 270);
    drawstring(footer);
    closepicture;
    setWpic(wind, graphpic);
  end;
end.
```

```
{ This unit contains all the code for the help system. The method of displaying help is    }
{ in lists inside dialog boxes (Similar to MSword). For the help system to work the       }
{ file containing the help data (filename specified by helpfilename) must be in the same  }
{ folder as the application.}

unit help;
interface
  uses
    MacPrint, XTTypeDefs, ListManager, XTTypeDefs2, Extender1, Extender2, mytypes;

  procedure initialise_help;

  procedure dohelp;


implementation

  const
    helpfilename = 'Analyzer help';     { name of the help file }
    helpButton = 1;                     { help button in topics dialog box }
    finishedbutton = 2;                 { finished button in topics dialog box }
    help_analyze = 1;                   { first entry in help file }
    help_pie_graph = 6;                 { last entry in help file }
    helpFileErr = 17653;                { resource number of help file error dialog box }
    TopicDialog = 258;                  { resource number of topics dialog box }
    Help_on_Topic_Dialog = 23151;       { resource number of help on topic dialog box }

  type
    topic = record
        index : integer;                { index into file for each help topic }
        title : string;
      end;

  var
    help_index : array[help_analyze..help_pie_graph] of topic;
    help_file : text;
    vRefNum : integer;

{-----------------------------------initialise_help-------------------------------------- }
{ Checks to see if the help file is available and if it is it initialises the topic names  }
{ and locations in the file. If the file is not available it displays a alert with an      }
{appropriate message. The key for the start of a topic in the help file is a numeric      }
{ character as the first character of a line.                                             }
  procedure initialise_help;
    var
      index, linecount : integer;
      ch : char;
      i : integer;
      helpList : string;
      vName : StringPtr;
      fndrInfo : FInfo;
      err : OSerr;

  begin
    for index := 1 to help_pie_graph do
      help_index[index].index := 0;
    err := GetVol(vName, vRefNum);
    err := GetFInfo(HelpFileName, vRefNum, fndrInfo);
```

```
    if err = NoErr then
      begin
        open(help_file, HelpFileName);
        reset(help_file);
        readln(help_file, helpList);
        for index := 1 to help_pie_graph do
          begin
            help_index[index].title := copy(helplist, 1, pos(';', helplist) - 1);
            helplist := copy(helplist, pos(';', helplist) + 1, length(helplist));
          end;
        index := 1;
        linecount := 1;
        while ((not (eof(help_file))) and (index <= help_pie_graph)) do
          begin
            if not eoln(help_file) then
              begin
                read(help_file, ch);
                if ('0' <= ch) and (ch <= '9') then
                  begin
                    help_index[index].index := linecount;
                    index := index + 1;
                  end;
              end;
            readln(help_file);
            linecount := linecount + 1
          end;
        close(help_file);
      end
    else
      begin
        sysbeep(30);
        i := Alert(helpFileErr, nil);
      end;
  end;

{-----------------------------------drawTopicList-------------------------------}
{ Calls LDraw for each of the cells in the list. Called to draw the list in the current }
{ graphport.                                                                      }
  procedure drawTopicList (numTopics : integer;
                           thelist : listHandle;
                           disprect : rect);
    var
      i : integer;
      tempcell : cell;
  begin
    for i := 0 to numtopics - 1 do
      begin
        setpt(tempcell, 0, i);
        LDraw(tempcell, thelist);
      end;
    framerect(disprect);
  end;


{-----------------------------------do_help_topic-------------------------------}
{ Creates a list and for each line in the help file, on the requested subject, puts a }
{ cell in the list and puts the line in it. The help on a topic continues until a blank }
{ line is encountered. This list is diplayed in a dialog box and is scrollable etc. When}
```

```
{ the finished buttom is clicked the dialog box is destroyed.                    }
  procedure do_help_topic (whichTopic : integer);
    const
      listpart = 1;
    var
      theList : ListHandle;
      cellSize, loc : Point;
      DP : DialogPtr;
      dialogitem : integer;
      itemhand : handle;
      itemnum : integer;
      tempcell : cell;
      disprect, rview, databounds : rect;
      CurrentRow, i : integer;
      DClick : boolean;
      data : Str255;
  begin
    DP := GetNewDialog(Help_on_Topic_Dialog, nil, pointer(-1));
    setport(DP);
    showwindow(DP);
    getditem(dp, listpart, itemnum, itemhand, disprect);
    rview := disprect;
    with rview do
      begin
        right := right - 16;
        SetPt(cellSize, right - left, 16);
        bottom := bottom - (bottom - top) mod cellsize.v;
      end;
    with disprect do
      setrect(disprect, left - 1, top - 1, right, (bottom - (bottom - top) mod cellsize.v + 1));
    setrect(databounds, 0, 0, 1, 0);

    TheList := LNew(rview, databounds, cellsize, 0, DP, false, false, false, true);
    currentrow := 0;
    reset(help_file);
    for i := 1 to help_index[whichTopic].index do
      readln(help_file);
    while not eoln(help_file) do
      begin
        setpt(tempcell, 0, currentrow);
        currentrow := currentrow + 1;
        i := laddrow(1, currentrow, thelist);
        readln(help_file, data);
        LsetCell(pointer(ord(@data) + 1), length(data), tempcell, thelist);
      end;
    LdoDraw(true, thelist);
    setWlist(DP, thelist);
    drawtopiclist(currentrow, thelist, disprect);
    LautoScroll(theList);
    thelist^^.selflags := lonlyone;
    ShowWindow(DP);
    repeat
      ModalDialog(nil, dialogitem);
      if dialogitem = 1 then
        begin
          getMouse(loc);
          Dclick := LClick(loc, Event.modifiers, thelist);
        end;
```

```
    until (dialogitem = finishedButton);
    disposdialog(DP);
  end;


{-----------------------------------dohelp-----------------------------------}
{ This procedure is called if the help botton is clicked in the about Analyzer box.  }
{  It displays, in a dialog box, a list of the topics on which help is available. If an  }
{  item in the list is double clicked or selected and the help button selected then  }
{ do_help_topic is called on that item. This is continued until finished is selected.  }
  procedure dohelp;
    const
      listpart = 5;
    var
      theList : ListHandle;
      tempStr : Str255;
      DP : DialogPtr;
      dialogitem : integer;
      data : str255;
      tempcell : cell;
      i : integer;
      err : OSerr;
      DClick : boolean;
      fndrInfo : FInfo;
      disprect : rect;
      loc : Point;

{ Finds the array index value which matches the string. }
    function findtopic (which : string) : integer;
      var
        foundnum : integer;
        found : boolean;
    begin
      found := false;
      foundnum := 1;
      while not found do
        if (help_index[foundnum].title = which) then
          found := true
        else
          foundnum := foundnum + 1;
      findtopic := foundnum;
    end;


{ Sets up the list of topics,as contained in the array help_index. }
    procedure set_up_list (var thelist : ListHandle;
                           DP : DialogPtr;
                           var disprect : rect);
      var

        itemhand : handle;
        itemnum : integer;
        rview, databounds : rect;
        cellSize : Point;
        i : integer;
    begin
      getditem(DP, listpart, itemnum, itemhand, disprect);
      rview := disprect;
```

```
    with rview do
      right := right - 16;
    with disprect do
      setrect(disprect, left - 1, top - 1, right, bottom + 1);
    setrect(databounds, 0, 0, 1, 0);
    SetPt(cellSize, 0, 0);
    TheList := LNew(rview, databounds, cellsize, 0, DP, false, false, false, true);
    LautoScroll(theList);
    i := laddrow(help_pie_graph, 0, thelist);
    for i := 0 to help_pie_graph - 1 do
      begin
        setpt(tempcell, 0, i);
        data := help_index[i + 1].title;
        LsetCell(pointer(ord(@data) + 1), length(data), tempcell, thelist);
      end;
    LdoDraw(true, thelist);
  end;


begin
  if help_index[1].index > 0 then
    begin
      err := setvol(nil, vRefNum);
      err := GetFInfo(HelpFileName, vRefNum, fndrInfo);
      if err = NoErr then
        begin
          open(Help_File, HelpFileName);
          DP := GetNewDialog(TopicDialog, nil, pointer(-1));
          setport(DP);
          showwindow(DP);
          set_up_list(thelist, DP, disprect);
          setWlist(DP, thelist);
          drawtopiclist(help_pie_graph, thelist, disprect);
          thelist^^.selflags := lonlyone;
          setpt(tempcell, 0, 0);
          LsetSelect(true, tempcell, thelist);
          repeat
            dialogitem := -1;
            ShowWindow(DP);
            repeat
              ModalDialog(nil, dialogitem);
              DClick := false;
              if dialogitem = Listpart then
                begin
                  getMouse(loc);
                  DClick := LClick(loc, Event.modifiers, thelist);
                end;
            until ((dialogitem = helpButton) or (dialogitem = finishedButton) or DClick);
            if ((dialogitem = helpButton) or DClick) then
              begin
                LMGetSelString(theList, tempStr);      { Get the selected string }
                if length(tempStr) > 0 then
                  begin
                    Do_Help_Topic(findtopic(tempstr));
                    setport(DP);
                    drawtopiclist(help_pie_graph, thelist, disprect);
                  end;
              end;
```

```
            until (dialogitem = finishedButton);
            disposdialog(DP);
            close(help_file)
          end
        else
          i := Alert(helpFileErr, nil);
      end
    else
      i := Alert(helpFileErr, nil);
  end;
end.
```

```
{ This unit contains the routines to analyse a file. It creates the files and write the }
{ results to them. It also calls the appropriate statistics procedures to keep the    }
{ stats updated. All the stats are added to the dte and dce global variables.         }


unit Panalyze;

interface
  uses
    XTTypeDefs, Extender1, mytypes, graphs;

  function get_file_name (pathname : string) : string;

  procedure createlookup;

  procedure analyze (var filename : string;
                     var char_per_sec : integer);

implementation


  const
    none = -1;          { do not know what sort of data being read at the moment }
    data = 0;           { currently reading data characters }
    fill = 1;           { currently reading fill characters }
    Layer_2 = 2;        { currently reading a layer 2 frame header or frame }
    Layer_3 = 3;        {currently reading a layer 3 packet header or packet }

  var
    testfile, savefile : text;                    { input and output text files }
    lookup : array['!'..'~', '!'..'~'] of integer;  { mnemonic conversion array }
    waitcursor : array[0..3] of CursHandle;       { ptr to a cursor record }
    validfiles : boolean;     { flag to indicate if the the files selected are correct }

{-------------------------------Convert-------------------------------}
{ Function that takes two characters as input and returns an integer.  }
{ Treats the characters as hexidecimal digits and returns an intger containing }
{ their base ten value.                                                }
  function convert (ch : char) : integer;
    var
      i : integer;
  begin
    if (ch >= 'A') then
      i := ord(ch) - ord('A') + 10
    else
      i := ord(ch) - ord('0');
    convert := i;
  end;

{-----------------------------TableLookUp-----------------------------}
{ Function takes two characters of a mnemonic and returns the corresponding }
{ ASCII integer value of that mnemonic.                                }
  function tablelookup (secch, firstch : char) : integer;
    var
      value : integer;
  begin
    value := lookup[firstch, secch];
    if (value <> -1) then
```

```
      tablelookup := value
    else
      tablelookup := convert(firstch) * 16 + convert(secch);
  end;

{-----------------------------------Get_Char_Rate-----------------------------------}
{ Dispalys the dialog box for obtaining a bits/sec rate from the user.              }
{ This is then converted into a characters / sec rate .                             }
  function get_char_rate : integer;
    const
      OK = 1;
    var
      DP : DialogPtr;
      dialogItem, Olditem : integer;
      fRefNum : integer;
  begin
    olditem := 2;
    DP := GetNewDialog(24145, nil, pointer(-1));
    selectwindow(DP);
    setport(DP);
    showwindow(DP);
    checkDitem(DP, Olditem);
    repeat
      ModalDialog(nil, dialogitem);
      if dialogitem <> OK then
        begin
          checkDitem(DP, Olditem);
          checkDitem(DP, dialogitem);
          olditem := dialogitem;
        end;
    until dialogitem = OK;
    disposDialog(DP);
    case Olditem of
      2 :
        get_char_rate := 150;
      3 :
        get_char_rate := 300;
      4 :
        get_char_rate := 600;
      5 :
        get_char_rate := 1200;
      6 :
        get_char_rate := 1;
    end;
  end;

{-----------------------------CreateLookUp-----------------------------}
{ Initialize the array to contain all -1 to start with (error value if returned) and }
{ then sets the values of the mnemonic pairs which exist.                           }
  procedure createlookup;
    var
      index1, index2 : char;
  begin
    for index1 := '!' to '~' do
      for index2 := '!' to '~' do
        lookup[index1, index2] := -1;

    lookup['n', 'u'] := 0;
```

```
    lookup['s', 'h'] := 1;
    lookup['s', 'x'] := 2;
    lookup['e', 'x'] := 3;
    lookup['e', 't'] := 4;
    lookup['e', 'q'] := 5;
    lookup['a', 'k'] := 6;
    lookup['b', 'l'] := 7;
    lookup['b', 's'] := 8;
    lookup['h', 't'] := 9;
    lookup['l', 'f'] := 10;
    lookup['v', 't'] := 11;
    lookup['f', 'f'] := 12;
    lookup['c', 'r'] := 13;
    lookup['s', 'o'] := 14;
    lookup['s', 'i'] := 15;
    lookup['d', 'l'] := 16;
    lookup['d', '1'] := -1;
    lookup['d', '2'] := -1;
    lookup['d', '3'] := -1;
    lookup['d', '4'] := -1;
    lookup['n', 'k'] := 21;
    lookup['s', 'y'] := 22;
    lookup['e', 'b'] := 23;
    lookup['c', 'n'] := 24;
    lookup['e', 'm'] := 25;
    lookup['s', 'b'] := 26;
    lookup['e', 'c'] := 27;
    lookup['f', 's'] := 28;
    lookup['g', 's'] := 29;
    lookup['r', 's'] := 30;
    lookup['u', 's'] := 31;
    lookup['s', 'p'] := 32;
  end;

{-------------------------------Get_File_Name-------------------------------- }
{ Extracts the filename from the full pathname of the file.                    }
  function get_file_name;
    var
      position : integer;
  begin
    position := 0;
    repeat
      delete(pathname, 1, position);
      position := pos(':', pathname);
    until position = 0;
    get_file_name := pathname;
  end;

{-----------------------------Standard_Format_File--------------------------- }
{ Checks the three strings against the three strings expected at the start of a }
{ standard format file as obtained from the comstate with a print of the data  }
{ buffer.                                                                      }
  function standard_format_file (line1, line2, line3 : string) : boolean;
  begin
    standard_format_file := false;
    if (line1 = ' ') and (line2 = ' ') then
      if (copy(line3, 1, 5) = 'ASCII') then
        standard_format_file := true
```

```
  end;

{-------------------------------SetUpFiles-------------------------------}
{ Displays a prompt to tell the use to open a input file and then displays the  }
{ standard dialog box for opening files. The file selected is checked for format  }
{ and if correct the files to write results to are created.                       }
  procedure setupfiles (var done : boolean;
                        var output_text_file : string;
                        var input_text_file : string);
    var
      line1, line2, line3 : string;
      defaultname : string;
      dialogitem : integer;
  begin
    done := false;
    input_text_file := '';
    output_text_file := '';
    input_text_file := OldFileName('open text file');
    if input_text_file <> '' then
      begin
        reset(testfile, input_text_file);
        readln(testfile, line1);          { remove header of printfile and blank lines }
        readln(testfile, line2);
        readln(testfile, line3);
        if standard_format_file(line1, line2, line3) then
          begin
            defaultname := concat(get_file_name(input_text_file), ' Res');
            output_text_file := newfilename('File to write results to', defaultname);
            if output_text_file <> '' then
              begin
                rewrite(savefile, output_text_file);
                rewrite(dte.graph, concat(output_text_file, 'dte.sp'));
                rewrite(dce.graph, concat(output_text_file, 'dce.sp'));
                done := true
              end
            else
              close(testfile)
          end
        else
          begin
            close(testfile);
            dialogitem := Alert(30381, nil);
          end;
      end;
  end;



{-----------------------Init_Packet_Counts-----------------------}

  procedure init_packet_counts (var thisone : packetype);
  begin
    with thisone do
      begin
        data := 0;
        rrpack := 0;
        rnrpack := 0;
```

```
        rejpack := 0;
        other := 0;
      end;
  end;

{-----------------------------------SetUp-----------------------------------}
{ Initializes the statstruct passed to it to the initial values.            }
  procedure setup (var present : statstruct);
  begin
    with present do
      begin
        frametype := -1;
        fillcount := 0;
        new(logicalchannel);
        logicalchannel^.next := nil;
        logicalchannel^.lcn_number := -1;
        logicalchannel^.charactercount := 0;
        init_packet_counts(logicalchannel^.L3packets);
        this_lcn := nil;
        init_packet_counts(L2packets);
        controlcount := 0;
        oschars := 0;
        headcount := 0;
        presentlyin := none;
        interpacket := 0;
        thispacket := 0;
        thisfill := 0;
      end;
  end;

{-------------------------------GatheringStats------------------------------}
{ Gathers the statistics of character counts from each of the logical channels and }
{ returns them in chcount.                                                  }
  procedure gatheringstats (var present : statstruct);
    var
      this : nextptr;
  begin
    this := present.logicalchannel;
    present.datacount := 0;
    while (this^.next <> nil) do
      begin
        present.datacount := this^.charactercount + present.datacount;
        this := this^.next;
      end;
  end;

{---------------------------------WriteOut----------------------------------}
{ Writes out the statistics gathered from analyzing the input file to the output }
{ file selected.                                                            }
  procedure writeout (var present : statstruct);
    const
      align = 45;
    var
      percent : real;
      next_lcn : nextptr;
      c : char;

    function leftjust (intext : string) : string;
```

```
    var
      i : integer;
  begin
    for i := length(intext) to align do
      intext := concat(intext, ' ');
    leftjust := intext;
  end;

  procedure writeoutlcn (this : nextptr);
  begin
    with this^ do
      begin
        writeln(savefile, leftjust('Logical channel number'), '=', lcn_number : 6);
        writeln(savefile, leftjust('Logical channel data character count'), '=', charactercount : 6);
        with L3packets do
          begin
            writeln(savefile, leftjust('Logical channel layer 3 data packet count'), '=', data : 6);
            writeln(savefile, leftjust('Logical channel layer 3 rr packet count'), '=', rrpack : 6);
            writeln(savefile, leftjust('Logical channel layer 3 rnr packet count'), '=', rnrpack : 6);
            writeln(savefile, leftjust('Logical channel layer 3 other packet count'), '=', other : 6);
          end;
      end;
  end;

begin

  gatheringstats(present);
  with present do
    begin
      writeln(savefile, leftjust('send fill count'), '=', fillcount : 6);
      writeln(savefile, leftjust('send control count'), '=', controlcount : 6);
      writeln(savefile, leftjust('send user data count'), '=', datacount : 6);
      writeln(savefile, leftjust('send L2 data packet count'), '=', L2packets.data : 6);
      writeln(savefile, leftjust('send L2 rr packet count'), '=', L2packets.rrpack : 6);
      writeln(savefile, leftjust('send L2 rnr packet count'), '=', L2packets.rnrpack : 6);
      writeln(savefile, leftjust('send L2 rej packet count'), '=', L2packets.rejpack : 6);
      writeln(savefile, leftjust('send L2 other packet count'), '=', L2packets.other : 6);
      percent := (fillcount / (datacount + controlcount + fillcount)) * 100;
      writeln(savefile, leftjust('Percentage of idle send line time'), '=', percent : 6 : 2, '%');
      writeln(savefile);
      next_lcn := logicalchannel;
      while (next_lcn^.next <> nil) do
        begin
          writeoutlcn(next_lcn);
          next_lcn := next_lcn^.next;
          writeln(savefile);
        end;
    end;
end;

{-----------------------------------Readin-----------------------------------}
{ This is the only procedure which reads from the input file. (Apart from the }
{ strings read to test if the file is in the standard input format, and the   }
{ readln's at the end of each line and removing blank lines between the dual line }
{ format (both in countup)). Readin is called by all other procedures which   }
{ require characters from the input file.                                     }
  procedure readin (var ch : char);
  begin
```

```
    read(testfile, ch);
  end;

{-------------------------Find_Lcn_Record-------------------------}
{ Searchs the linked list of lcn records for one which has the requested lcn }
{ number. If one does not exist it creates a new node and adds it to the end of }
{ the list. }
  procedure find_lcn_record (head : nextptr;
                             var current : nextptr;
                             this_lcn : integer);
    var
      found : boolean;
  begin
    found := false;
    current := head;
    while ((current^.next <> nil) and (not found)) do
      begin
        if (current^.lcn_number = this_lcn) then
          found := true
        else
          current := current^.next;
      end;
    if (not found) then
      begin
        new(current^.next);
        current^.next^.next := nil;
        current^.lcn_number := this_lcn;
        current^.charactercount := 0;
        init_packet_counts(current^.L3packets);
      end;
  end;

{-------------------------Layer2-------------------------}
{ Reads in and analyzes the layer two header of the packet. If the frame type }
{ indicates it contains data for layer three the procedure passes the packet on to }
{ layer3. If it is a layer two frame only then it is all read in . The appropriate }
{ statistics counters are incremented. }
  procedure layer2 (var thisone : statstruct);
    var
      finished : boolean;
      ch, lastch : char;
      lcn : integer;
  begin
    finished := false;
    lastch := ' ';
    with thisone do
      begin
        while ((not (eoln(testfile))) and (not finished)) do
          begin
            readin(ch);
            if (ch <> ' ') then
              oschars := oschars + 1;
            if ((ch = ' ') or (oschars = 2)) then
              begin
                controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
                headcount := headcount + ((oschars div 2) + (oschars mod 2));
                oschars := 0;
                interpacket := interpacket + 1;
```

```
                if ((headcount = 2) and (frametype = -1)) then
                  begin
                    if (ch <> ' ') then
                      frametype := tablelookup(ch, lastch)
                    else
                      frametype := ord(lastch);
                    if ((frametype mod 2) = 0) then
                      begin
                        presentlyin := layer_3;
                        L2packets.data := L2packets.data + 1;
                        finished := true
                      end
                    else
                      begin
                        case (frametype mod 32) of
                          17 :
                            L2packets.rejpack := L2packets.rejpack + 1;
                          5 :
                            L2packets.rnrpack := L2packets.rnrpack + 1;
                          1 :
                            L2packets.rrpack := L2packets.rrpack + 1;
                          otherwise
                            L2packets.other := L2packets.other + 1;
                        end;
                      end;
                    frametype := 0;
                  end;
              end;

            if ((lastch = ':') and (ch = 'G')) then
              begin
                controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
                interpacket := interpacket + ((oschars div 2) + (oschars mod 2));
                oschars := 0;
                presentlyin := fill;
                finished := true;
                headcount := 0;
                frametype := -1;
              end;
            lastch := ch;
          end;
      end;
  end;


{-------------------------------Layer3-------------------------------}
{ Reads in and analyzes the layer three header of the packet. If the frame type }
{ indicates it is a data packet the procedure passes the packet on to readdata. }
{ If it is a layer three frame only then it is all read in. The appropriate statistics}
{ counters are incremented.                                          }
  procedure layer3 (var thisone : statstruct);
    var
      finished : boolean;
      ch, lastch : char;
      lcn : integer;
  begin
    finished := false;
```

```
    lastch := ' ';
    with thisone do
      begin
        while ((not (eoln(testfile))) and (not finished)) do
          begin
            readin(ch);
            if (ch <> ' ') then
              oschars := oschars + 1;
            if ((ch = ' ') or (oschars = 2)) then
              begin
                controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
                headcount := headcount + ((oschars div 2) + (oschars mod 2));
                interpacket := interpacket + ((oschars div 2) + (oschars mod 2));
                oschars := 0;

                if ((headcount = 4) and (this_lcn = nil)) then
                  begin
                    if (ch <> ' ') then
                      lcn := tablelookup(ch, lastch)
                    else
                      lcn := ord(lastch);
                    find_lcn_record(logicalchannel, this_lcn, lcn);
                  end;

                if ((headcount = 5) and (frametype = 0)) then
                  begin
                    if (ch <> ' ') then
                      frametype := tablelookup(ch, lastch)
                    else
                      frametype := ord(lastch);
                    if ((frametype mod 2) = 0) then
                      begin
                        presentlyin := data;
                        headcount := 0;
                        oschars := 0;
                        finished := true;
                        this_lcn^.L3packets.data := this_lcn^.L3packets.data + 1;
                      end
                    else
                      begin
                        with this_lcn^.L3packets do
                          begin
                            frametype := frametype mod 32;
                            case frametype of
                              5 :
                                rnrpack := rnrpack + 1;
                              1 :
                                rrpack := rrpack + 1;
                              otherwise
                                other := other + 1;
                            end;
                          end;
                      end;
                  end;
              end;

            if ((lastch = ':') and (ch = 'G')) then
              begin
```

```
                controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
                interpacket := interpacket + ((oschars div 2) + (oschars mod 2));
                oschars := 0;
                presentlyin := fill;
                finished := true;
                headcount := 0;
                frametype := -1;
                this_lcn := nil;
              end;
            lastch := ch;
          end;
      end;
  end;


{-------------------------------ReadData-------------------------------}
{ Reads in and counts up the layer three data of the packet. The histograms and }
{ file of inter packet times and lengts are updated after all the packet has been }
{ read.                                                                }
  procedure readdata (var thisone : statstruct);
    var
      finished : boolean;
      ch, lastch : char;
      numch : integer;
  begin
    finished := false;
    with thisone do
      begin
        while ((not (eoln(testfile))) and (not finished)) do
          begin
            readin(ch);
            if (ch <> ' ') then
              oschars := oschars + 1;
            if (ch = ' ') then
              begin
                numch := (oschars div 2) + (oschars mod 2);
                this_lcn^.charactercount := this_lcn^.charactercount + numch;
                thispacket := thispacket + numch;
                oschars := 0;
              end;
            if ((lastch = ':') and (ch = 'G')) then
              begin
                if (oschars = 2) then
                  begin
                    controlcount := controlcount + 2;        { add layer 2 trailer }
                    this_lcn^.charactercount := This_lcn^.charactercount - 1;
                    thispacket := thispacket - 1             { remove layer 2 trailer }
                  end
                else
                  controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
                oschars := 0;
                presentlyin := fill;
                finished := true;
                frametype := -1;
                this_lcn := nil;
                writeln(graph, interpacket, chr(9), thispacket);
                updatehist(phist, thispacket);
                updatehist(ihist, interpacket);
```

```
                thispacket := 0;
                interpacket := 0;
              end;
            lastch := ch;
          end;
      end;
  end;


{-------------------------------ReadFill-------------------------------- }
{ Reads in and counts up the idle fill characters between packets. If a non-fill  }
{ character is encountered then the procedure terminates and control passes       }
{  to layer 2 procedure.                                                          }
  procedure readfill (var thisone : statstruct);
    var
      finished : boolean;
      ch, lastch : char;
      i : integer;
  begin
    ch := ' ';
    lastch := ' ';
    finished := false;
    with thisone do
      begin
        while ((not (eoln(testfile))) and (not finished)) do
          begin
            readin(ch);
            if (ch = '.') then
              thisfill := thisfill + 1
            else
              begin
                if (ch <> ' ') then
                  oschars := oschars + 1;
                if ((lastch <> ' ') and (lastch <> '.')) then
                  begin
                    finished := true;
                    interpacket := interpacket + round(thisfill / 3);
                    fillcount := fillcount + round(thisfill / 3);
                    thisfill := 0;
                    presentlyin := layer_2;
                  end;
              end;
            lastch := ch;
          end;
      end;
  end;

{-----------------------------ReadStart---------------------------------- }
{ Reads in and throws away characters until what part of a sample is currently   }
{ being read. This the state the program starts in for each direction of each sample }
{analysed.                                                                        }
  procedure readstart (var thisone : statstruct);
    var
      finished : boolean;
      ch, lastch : char;
  begin
    ch := ' ';
    lastch := ' ';
```

```
    finished := false;
    with thisone do
      begin
        while ((not (eoln(testfile))) and (not finished)) do
          begin
            readin(ch);
            if (ch = '.') and (lastch = '.') then
              begin
                finished := true;
                presentlyin := fill
              end
            else if (ch = 'G') and (lastch = ':') then
              begin
                finished := true;
                presentlyin := fill
              end;
            lastch := ch;
          end;
      end;
  end;

{-----------------------------------Countup-----------------------------------}
{ Selects the procdure that should be currently active on the value of presentlyin }
  procedure countup (var present : statstruct);
  begin
    with present do
      begin
        while not (eoln(testfile)) do
          begin
            case presentlyin of
              none :
                readstart(present);
              fill :
                readfill(present);
              layer_2 :
                layer2(present);
              data :
                readdata(present);
              layer_3 :
                layer3(present);
            end;
          end;
        readln(testfile);
      end;
  end;

{-------------------------Analyze-------------------------------}
{ Sets up the files required and initialises all the statistics counters. }
{ Loops around analyzing the file until EOF. }
{ Writes out the results to the file selected. }
  procedure analyze;
    var
      input_filename : string;
      i : integer;
  begin
    setupfiles(validfiles, filename, input_filename);
    if validfiles then
      begin
```

```
        char_per_sec := get_char_rate;
        for i := 0 to 3 do
          waitcursor[i] := getcursor(i + 512);
        setcursor(waitcursor[0]^^);
        setup(dte);                { Initialize the statistics counters }
        inithistogram(dte.phist, 0, 256, 20, 'DTE Data Packet Length', 'Data packet length (chars)');
        inithistogram(dte.ihist, 0, 600, 20, 'DTE Inter Packet Time', 'Inter packet time (sec)');
        writeln(dte.graph, 'Interpacket', chr(9), 'Packet Length');
        setup(dce);
        inithistogram(dce.phist, 0, 256, 20, 'DCE Data Packet Length', 'Data packet length (chars)');
        inithistogram(dce.ihist, 0, 600, 20, 'DCE Inter Packet Time', 'Inter packet time (sec)');
        writeln(dce.graph, 'Interpacket', chr(9), 'Packet Length');
        writeln(savefile, 'Results of analyzing ', input_filename);
        i := 0;

        while (not (eof(testfile))) do
          begin
            countup(dte);                { one line for each direction }
            i := (i + 1) mod 4;
            setcursor(waitcursor[i]^^);
            countup(dce);
            i := (i + 1) mod 4;
            setcursor(waitcursor[i]^^);
            readln(testfile);            { remove blank line between dual lines }
          end;

        writeln(savefile, 'DTE Results');    { Print out the results }
        writeout(dte);
        writeln(savefile);
        writeln(savefile, 'DCE Results');
        writeout(dce);
        close(testfile);
        close(savefile);
        writeln(dte.graph);
        writeln(dce.graph);
        close(dte.graph);
        close(dce.graph);
        setcursor(arrow);
      end;
  end;
end.
```

```
program Analyser (input, output);
  uses
    XTTypedefs, Extender1, MacPrint, ListManager, XTTypeDefs2, mytypes;

  type
    window = record
        wind : windowptr;
        WR : WindowRecord;
        Drawn : Boolean;
      end;


  var

    oldfilename, outputfilename : string;
    applemenu, filemenu, editmenu, runmenu, windmenu : menuhandle;
    Icursor : CursHandle;
    graphwind : array[1..7] of window;
    drect, vrect, wrect : rect;
    reply : sfreply;
    err : OSerr;
    stylechange : boolean;
    hPrint : THPrint;
    defaultname : str255;
    currentwind : windowptr;
    i : integer;
    char_sec : integer;
    fontsize : sizelist;
    dterep, dcerep : replications;

{From Extender2}
  procedure XTsetfont (wptr : WindowPtr;
                       fontName : Str255);
  external;
  procedure DoSizeMenu (wptr : WindowPtr;
                       itemnum : integer;
                       theSizes : SizeList);
  external;
  function newprinthandle : THPrint;
  external;
  function saveWPic (wptr : WindowPtr;
                       var reply : SFReply) : OSErr;
  external;
  function TEPrint (TEH : TEHandle;
                       hPrint : THPrint) : OSErr;
  external;
  function PrintPic (thePic : picHandle;
                       hPrint : THPrint) : OSErr;
  external;

  function BitstoPic (theBits : BitMap) : picHandle;
  external;

  function getWpic (wptr : WindowPtr) : picHandle;
  external;


{ From NProcAnalyse Unit }
```

```
  function get_file_name (pathname : string) : string;
  external;

  procedure createlookup;
  external;

  procedure analyze (var filename : string;
                     var char_per_sec : integer);
  external;


{ From help unit }
  procedure initialise_help;
  external;

  procedure dohelp;
  external;


{from graph unit}
  function int_to_string (i : integer) : string;
  external;

  function real_to_string (num : real) : string;
  external;

  function TalliedSigmaQ (tally : atallytype) : real;
  external;

  procedure inittally (var tally : atallytype);
  external;

  procedure updatetally (var tally : atallytype;
                         obs : real);
  external;

  procedure showhistogram (var wind : windowptr;
                           histo : ahistogramtype;
                           xcoord, ycoord, charsec : integer;
                           footer : string);
  external;

  procedure piegraph (fill, control : integer;
                      this : statstruct;
                      var wind : windowptr;
                      title : string;
                      footer : string);
  external;


{----------Set_up_menus----------}
  procedure set_up_menus;
  begin
    Stdmenus(applemenu, filemenu, editmenu);
    setmenuitem(applemenu, 1, 'About Analyzer');
    enableitem(applemenu, 1);
    setmenuitem(filemenu, 1, 'New Text Wind');
    setmenuitem(filemenu, 4, 'Close Text Wind');
```

```
    Enableallitems(filemenu, true);
    Enableallitems(editmenu, true);
    disableitem(editmenu, 1);
    disableitem(editmenu, 8);
    disableitem(filemenu, 1);
    disableitem(filemenu, 7);
    runmenu := BuildMenu(runId, 'RUN', 'Analyse;Analyse next in series;Summary of series');
    disableitem(runmenu, 2);
    disableitem(runmenu, 3);
    windmenu := BuildMenu(windId, 'Windows', 'Text Wind;DTE graph pack;DCE graph pack;DTE inter;DCE
      inter;DTEpie;DCEpie');
    enableallitems(windmenu, false);
    enableitem(windmenu, 1);
  end;

{----------------------------Create_Windows---------------------------------}
{ Create the analyser text window and set all the other window pointers to nil  }
{ to indicate that they have not been used yet.                                 }
  procedure create_windows;
    var
      i : integer;
      pictrect : rect;
  begin
    setrect(drect, 10, 10, 470, 270);
    setrect(vrect, 10, 10, 470, 270);
    setrect(wrect, 3, 40, 510, 335);
    setrect(pictrect, 0, 0, 1000, 1000);
    graphwind[1].wind := createwindow(graphwind[1].WR, wrect, 'Analyser Text Window', 4, true, false, true, true,
      false);
    TEMakeNew(graphwind[1].wind, drect, vrect);
    for i := 2 to 7 do
      graphwind[i].wind := nil;
  end;

{-----------------------------------do_about_analyzer---------------------------- }
{ Display the about analyser dialog box and call the help procedures if the help  }
{ button is pushed.                                                               }
  procedure do_about_analyzer;
    const
      helpbutton = 5;
    var
      dialogItem : integer;
  begin
    dialogitem := Alert(145, nil);
    if dialogitem = helpButton then
      dohelp;
  end;


  function condense (title : string) : string;
  begin
    condense := concat(copy(title, 1, 3), copy(title, 11, 3));
  end;


{-------------------------mark_wind---------------------------------------- }
{ Place a tick beside the entry in the Window menu that is the current window. }
  procedure mark_wind;
```

```
    var
      i : integer;
  begin
    for i := 1 to 7 do
      begin
        markmenuitem(windmenu, i, ' ');
        if frontwindow = graphwind[i].wind then
          markmenuitem(windmenu, i, chr(18));
      end;
    if frontwindow = graphwind[1].wind then
      begin
        enableallitems(editmenu, true);
        disableitem(editmenu, 1);
        disableitem(editmenu, 8);
        enableitem(filemenu, 2);
        enableitem(filemenu, 4)
      end
    else
      begin
        disableitem(filemenu, 2);
        disableitem(filemenu, 4);
        enableallitems(editmenu, false);
      end;
  end;

{-------------------------------draw_graphs------------------------------- }
{ If a graph window is selected and the graph wasn't been drawn on it yet then  }
{ draw it. Then display the window.                                          }
  procedure draw_graphs (windnum : integer);
    var
      grect : rect;
  begin
    setrect(grect, 3, 40, 510, 335);
    case windnum of
      2 :
        begin
          if graphwind[2].wind = nil then
            graphwind[2].wind := createwindow(graphwind[2].WR, grect, 'Dte Graph packet size', 4, true, false, true,
       true, false);
          if graphwind[2].drawn = false then
            showhistogram(graphwind[2].wind, dte.phist, 100, 200, 1, dtefoot);
        end;
      3 :
        begin
          if graphwind[3].wind = nil then
            graphwind[3].wind := createwindow(graphwind[3].WR, grect, 'Dce Graph packet size', 4, true, false, true,
       true, false);
          if graphwind[3].drawn = false then
            showhistogram(graphwind[3].wind, dce.phist, 100, 200, 1, dcefoot);
        end;
      4 :
        begin
          if graphwind[4].wind = nil then
            graphwind[4].wind := createwindow(graphwind[4].WR, grect, 'Dte Graph interpacket', 4, true, false, true,
       true, false);
          if graphwind[4].drawn = false then
            showhistogram(graphwind[4].wind, dte.ihist, 100, 200, char_sec, dtefoot);
        end;
```

```
      5 :
        begin
          if graphwind[5].wind = nil then
            graphwind[5].wind := createwindow(graphwind[5].WR, grect, 'Dce Graph interpacket', 4, true, false, true,
       true, false);
          if graphwind[5].drawn = false then
            showhistogram(graphwind[5].wind, dce.ihist, 100, 200, char_sec, dcefoot);
        end;
      6 :
        begin
          if graphwind[6].wind = nil then
            graphwind[6].wind := createwindow(graphwind[6].WR, grect, 'Dte Char  pie chart', 4, true, false, true,
       true, false);
          if graphwind[6].drawn = false then
            piegraph(dte.fillcount, dte.controlcount, dte, graphwind[6].wind, pieheader, dtefoot);
        end;
      7 :
        begin
          if graphwind[7].wind = nil then
            graphwind[7].wind := createwindow(graphwind[7].WR, grect, 'Dce Char  pie chart', 4, true, false, true,
       true, false);
          if graphwind[7].drawn = false then
            piegraph(dce.fillcount, dce.controlcount, dce, graphwind[7].wind, pieheader, dcefoot);
        end;
    end;
    graphwind[windnum].drawn := true;
  end;

{-------------------------updatereps---------------------------------- }
{ Update the tallys for the analysis of a series statistics.           }
  procedure updatereps (itemnum : integer;
                        var dterep, dcerep : replications);
    var
      dtetotal, dcetotal : integer;
  begin
    with dte do
      dtetotal := datacount + fillcount + controlcount;
    with dce do
      dcetotal := datacount + fillcount + controlcount;
    if itemnum = 1 then
      begin
        inittally(dterep.data);
        inittally(dcerep.data);
        inittally(dterep.fill);
        inittally(dcerep.fill);
        inittally(dterep.control);
        inittally(dcerep.control);
      end;
    updatetally(dterep.data, (dte.datacount / dtetotal));
    updatetally(dcerep.data, (dce.datacount / dcetotal));
    updatetally(dterep.fill, (dte.fillcount / dtetotal));
    updatetally(dcerep.fill, (dce.fillcount / dcetotal));
    updatetally(dterep.control, (dte.controlcount / dtetotal));
    updatetally(dcerep.control, (dce.controlcount / dcetotal));
  end;


{-------------------------------displayreps---------------------------- }
```

```
{display the dialog box conataining the results of the analysis of the series.  }
{If the "print screen button is selected a screen dump is done.                 }
  procedure displayreps (dterep, dcerep : replications);
    const
      OK = 23;
      Print = 24;
    var
      DP : DialogPtr;
      dialogitem : integer;
      err : OSerr;
      thebitmap : BitMap;
      itemHandl : Handle;
      disprect : rect;
      itemType : integer;

    procedure setvalue (field : integer;
                        value : real);
      var
        itemHandl : Handle;
        disprect : rect;
        itemType : integer;
    begin
      value := value * 100;
      GetDItem(DP, field, itemType, itemHandl, disprect);
      SetIText(itemHandl, real_to_string(value));
    end;

  begin
    DP := GetNewDialog(19969, nil, pointer(-1));
    with dterep do
      begin
        setvalue(11, data.integral / data.noofobs);
        setvalue(13, control.integral / control.noofobs);
        setvalue(15, fill.integral / fill.noofobs);
        setvalue(12, TalliedSigmaQ(data));
        setvalue(14, TalliedSigmaQ(control));
        setvalue(16, TalliedSigmaQ(fill));
      end;
    with dcerep do
      begin
        setvalue(17, data.integral / data.noofobs);
        setvalue(19, control.integral / control.noofobs);
        setvalue(21, fill.integral / fill.noofobs);
        setvalue(18, TalliedSigmaQ(data));
        setvalue(20, TalliedSigmaQ(control));
        setvalue(22, TalliedSigmaQ(fill));
      end;
    GetDItem(DP, 26, itemType, itemHandl, disprect);
    SetIText(itemHandl, int_to_string(dcerep.data.noofobs));
    setport(DP);
    showwindow(DP);
    repeat
      ModalDialog(nil, dialogitem);
      if dialogitem = Print then
        err := Printpic(BitsToPic(ScreenBits), hPrint);
    until (dialogitem = OK);
    disposdialog(DP);
  end;
```

```
{----------------------------Analyse_A_file---------------------------------- }
{ This is the procedure called if "Analyse" or "Analyse next in Series" is    }
{selected from the run menu. It calls the procedure Analyse to do the analysis }
{then displays the text results in the window.                                }
  procedure Analyse_A_file (itemnum : integer);
    var
      i : integer;
      WD : WData;
  begin
    oldfilename := outputfilename;
    setport(graphwind[1].wind);

    analyze(outputfilename, char_sec);

    if outputfilename <> '' then
      begin
        enableallitems(windmenu, true);
        enableitem(runmenu, 2);
        enableitem(runmenu, 3);
        for i := 2 to 7 do
          graphwind[i].drawn := false;
        if length(outputfilename) < 64 then
          begin
            reply.fname := outputfilename;
            reply.good := true;
            showwindow(graphwind[1].wind);
            disableitem(filemenu, 1);
            enableitem(windmenu, 1);
            KillWindow(graphwind[1].wind);
            graphwind[1].wind := createwindow(graphwind[1].WR, wrect, 'Analyser Text Window', 4, true, false,
     true, true, false);
            TEMakeNew(graphwind[1].wind, drect, vrect);
            XTsetfont(graphwind[1].wind, 'Monaco');
            DoSizeMenu(graphwind[1].wind, 1, fontsize);
            selectwindow(graphwind[1].wind);
            getWData(graphwind[1].wind, WD);
            TERecall(WD.TEH, reply);
            if reply.good then
              begin
                TEsetScrollRange(graphwind[1].wind);
                disableitem(filemenu, 2)
              end
          end;
        updatereps(itemnum, dterep, dcerep);
      end
    else
      outputfilename := oldfilename;
  end;


{---------------------- MAIN ----------------------------}

begin
  XTendInit;
  initialise_help;
  initfonts;
```

```
  Icursor := getCursor(1);
  createlookup;
  set_up_menus;
  create_windows;
  selectwindow(graphwind[1].wind);

  XTsetfont(graphwind[1].wind, 'Monaco');
  with fontsize do
    begin
      sizecount := 1;                    { Set up the font for the text window }
      thesize[1] := 9;
    end;
  DoSizeMenu(graphwind[1].wind, 1, fontsize);

  PROpen;
  hPrint := NewPrintHandle;
  err := TEFromScrap;
  repeat
    repeat
      systemtask;
      maintainCursor(arrow, Icursor^^, arrow);
    until XTGetNextEvent(EveryEvent, event);

    HandleEvent(event, Whathappened);
    mark_wind;
    reply.good := False;
    with whathappened do
      case menunum of
        runId :
          begin
            case itemnum of

              1, 2 :        {Analyse and  Analyse next in series }
                Analyse_A_File(itemnum);

              3 :           { Results of Series }
                displayreps(dterep, dcerep);
            end;
          end;
        windId :            { Select a window }
          begin
            if itemnum <> 1 then
              draw_graphs(itemnum);
            selectwindow(graphwind[itemnum].wind);
            mark_wind
          end;
        fileId :
          case Itemnum of

            1 :                 {New}
              begin
                showwindow(graphwind[1].wind);
                disableitem(filemenu, 1);
                enableitem(windmenu, 1);
              end;

            2 :                 {Open}
              begin
```

```
            KillWindow(graphwind[1].wind);
            graphwind[1].wind := createwindow(graphwind[1].WR, wrect, 'Analyser Text Window', 4, true,
false, true, true, false);
            TEMakeNew(graphwind[1].wind, drect, vrect);
            XTsetfont(graphwind[1].wind, 'Monaco');
            DoSizeMenu(graphwind[1].wind, 1, fontsize);
            TERecall(TEH, reply);
            if reply.good then
              begin
                TEsetScrollRange(graphwind[1].wind);
                disableitem(filemenu, 2);
              end;
            selectwindow(graphwind[1].wind);
          end;

        4 :                 {Close text wind}
          begin
            KillWindow(graphwind[1].wind);
            graphwind[1].wind := createwindow(graphwind[1].WR, wrect, 'Analyser Text Window', 4, true,
false, true, true, false);
            TEMakeNew(graphwind[1].wind, drect, vrect);
            XTsetfont(graphwind[1].wind, 'Monaco');
            DoSizeMenu(graphwind[1].wind, 1, fontsize);
            hidewindow(graphwind[1].wind);
            enableitem(filemenu, 1);
            disableitem(windmenu, 1);
          end;

        5, 6 :              {Save, SaveAs}
          if frontwindow = graphwind[1].wind then
            TESave(TEH, reply)
          else if frontwindow <> nil then
            begin
              reply.good := true;
              currentwind := frontwindow;
              getWtitle(currentwind, defaultname);
              defaultname := condense(defaultname);
              defaultname := concat(get_file_name(outputfilename), defaultname);
              reply.fname := newfilename('new pict file', defaultname);
              if length(reply.fname) > 0 then
                err := saveWPic(frontwindow, reply);{, 'MDRW', 'PICT'}
            end;

        9 :                 { Page Setup }
          stylechange := PrStlDialog(hPrint);

        10 :              { Print }
          if frontwindow = graphwind[1].wind then
            err := TEPrint(TEH, hPrint)
          else
            err := PrintPic(getWpic(frontwindow), hPrint);
        otherwise
          ;       { is automatic }
      end;
    EDITId :
      case ItemNum of {editMenu}
        3 :
          TECut(TEH);
```

```
            4 :
              TECopy(TEH);
            5 :
              TEPaste(TEH);
            6 :
              TEDelete(TEH);
            9 :
              TESelectAll(TEH);
            otherwise
              ;
          end;  {case}
        appleId :
          if itemnum = 1 then
            do_about_analyzer;
        otherwise
          ;
      end;
  until ExitRequest(WhatHappened);
  if (0 = ZeroScrap) then
    err := TEToScrap;
  for i := 1 to 7 do
    if graphwind[i].wind <> nil then
      killwindow(graphwind[i].wind)
end.
```

PC Code

The code for the PC was developed on a Exzel 286 lent to me by a classmate. Initial development was done in Turbo Pascal version 4 and switched to version 5 when version 4 was found to be lacking in features that were required.

The first printout is the help file.

The program is divided up into 3 units

Types Unit This contains the types and constants that are used globally in the program.

MyGraph Unit This contains the procedures for manipulation and graphing of the of the statistics.

Main This contains the code for the following :

1 The Analysis of a file containing a traffic sample.
2 The help system
3 The menu system.

1 Analyze : To analyze a traffic sample.
Prompts for the name of the file containing the traffic sample and then a file name to use to write the results to. Three files are created one with this name (main results)and two with it as their base (spread sheet files).
The Bit rate asked for :
This refers to the bit rate of the line the sample was captured from. If the "do not know option is chosen the inter packet times are in characters not in seconds.
The series question :
This is only asked on the second (and susequent) requests to analyse a file. If the answer is n or N then the tallies are reset and this saample being analysed is taken as the first in the series. If the answer is y or Y then this sample is added to the tallies that already exist.

2 Pie Graphs : The three segments represent the following.
Fill : This is the proportion of time the line is Idle and so being filled with idle characters
Control : This is the proportion of characters sent which are part of the protocol eg not user data.
Data : This is the proportion of characters sent which are user data.

3 Inter Packet times histogram :
This is a histogram of the time in seconds between packets containing user data. The scale along the horizontal axis changes with the bit rate selected. If the do not know selection is chosen then the horizontal axis is in characters between data packets and not seconds.
It is for all logical channels.

4 Packet Length histogram :
This is a histogram of the length, in characters, of data packets.
It is for all active logical channels.

5 Display a Text File :
Dispalys a text file by reading the file and writing it to the screen twenty lines at a time. A return is required from the user between sections of twenty lines to give the user time to read the file.

6 Results of Replication :
Displays the mean and standard deviation of a tally of the samples analysed in the series. The percentage of each type of charcter in each sample is added to the tally of that type and so the mean represents the mean percentage over all the samples.

7 Dos Shell :
Creates a DOS command shell to allow user to execute commands like dir and cd to find files and change directories, while still inside the program.
Exit returns user back to program.

```
{ This unit contains the global type and constant declarations.          }
UNIT typesunit;

INTERFACE

CONST
  MaxHistClasses = 40;
  MyMaxX = 400;
  MyMaxY = 300;
  dtefoot = 'DTE to DCE. One Way Traffic. All Logical Channels.';
  dcefoot = 'DCE to DTE. One Way Traffic. All Logical Channels.';
  dtePiehead =  'DTE Character Type Percentages';
  dcePiehead =  'DCE Character Type Percentages';

TYPE

 atallytype = record
   noOfObs : 0..maxint;
   integral, sumofsq : real;
   minobs, maxobs : real
  end;

 replication = record
    fill,data,control : atallytype;
    end;

 ahistogramtype = record
   title : string;
   hor_title : string;
   NoOfClasses : 0..MaxHistClasses;
  classwidth : real;
   low, high : integer;
   tally : atallytype;
   counters : array[0..MaxHistClasses] OF 0..Maxint;
   overflow, underflow : 0..Maxint
 end;

 lcnptr = ^datacount;
 datacount = record
    lcnnum : integer;
    count : integer;
    next : lcnptr;
    end;

IMPLEMENTATION

END.
```

```
{ This unit contains the graphing and statistics gathering routines }

UNIT mygraph;

INTERFACE
 USES
  crt,Graph,typesunit;

 FUNCTION int_to_string(i : integer) : string;

 FUNCTION real_to_string(num : real) : string;

 PROCEDURE inittally (var tally : atallytype);

 PROCEDURE updatetally(var tally : atallytype; obs : real);

 FUNCTION TalliedSigmaQ(tally : atallytype) : real;

 PROCEDURE inithistogram (var histo : ahistogramtype;
            lowp, highp : integer;
            classesp : integer;
            name : string;
            horzon : string);

 PROCEDURE updatehist (var histo : ahistogramtype;
            obs : integer);

 PROCEDURE showhistogram (
            histo : ahistogramtype;
            xcoord, ycoord : integer;
            speed : integer;
            footer : string);

 PROCEDURE showpiechart(fill,control : integer;
                lcn :lcnptr;
                title : string; foot : string);

IMPLEMENTATION
const
 y_axis_height = 150;
 x_axis_width = 200;


{ integer argument returns a string represntation of the integer }

FUNCTION int_to_string;
  var
    s : string;
 begin
  if i > 9 then
    begin
     s := concat(int_to_string(i div 10), chr((i mod 10) + ord('0')));
    end
  else
   s := chr(i + ord('0'));
  int_to_string := s;
 end;
```

```
{ take a real argument and returns a string cotaining the real number too      }
{ two decimal places.                                                          }

FUNCTION real_to_string;
   var
     temp : string;
 begin
   temp := concat(int_to_string(trunc(num)), '.');
   num := (num - trunc(num)) * 10;
   temp := concat(temp, int_to_string(trunc(num)));
   num := (num - trunc(num)) * 10;
   real_to_string := concat(temp, int_to_string(trunc(num)));
 end;


{ calculate standard deviation of the observations added to the tally. }
FUNCTION TalliedSigmaQ;
 begin
   with tally do
     if noofobs > 1 then
       TalliedSigmaQ := sqrt(abs((sumofsq - sqr(integral) / noofobs) / (noofobs - 1)))
     else
       TalliedSigmaQ := 0.0;
 end;


FUNCTION x(xcoord : integer) : integer;
 begin
   x := round((getmaxX / mymaxX) * xcoord)
 end;

FUNCTION y(ycoord : integer) : integer;
 begin
   y := round((getmaxy / mymaxY) * ycoord);
 end;


PROCEDURE inittally;
   const
     alargenumber = 20000;
 begin
   with tally do
     begin
       noofobs := 0;
       integral := 0.0;
       sumofsq := 0.0;
       minobs := alargenumber;
       maxobs := 0;
     end;
 end;
```

```
PROCEDURE updatetally ;
  begin
    with tally do
      begin
        noofobs := noofobs + 1;
        integral := integral + obs;
       sumofsq := sumofsq + sqr(obs);
        if obs < minobs then
          minobs := obs;
        if obs > maxobs then
          maxobs := obs;
      end;
  end;

PROCEDURE inithistogram;
  var index : 1..maxhistclasses;
  begin
    with histo do
      begin
       noofclasses := classesp;
       low := lowp;
       high := highp;
      classwidth := (high -low) / noofclasses;
       inittally(tally);
      for index := 1 to noofclasses do
        counters[index] := 0;
      overflow := 0;
      underflow := 0;
       title := name;
       hor_title := horzon;
      end;
  end;

{ add an observation to the histogram. }
PROCEDURE updatehist;
    var
      class : 1..maxhistclasses;
  begin
    updatetally(histo.tally, obs);
    if obs < histo.low then
     histo.underflow := histo.underflow + 1
    else
      if obs > histo.high then
       histo.overflow := histo.overflow + 1
      else
        begin
          class := trunc((obs - histo.low) / histo.classwidth) + 1;
          if class > histo.noofclasses then
            class := histo.noofclasses;
          histo.counters[class] := histo.counters[class] + 1;
        end
  end;
```

```
{-----------------------------------ShowHistogram-----------------------------}
(* Displays a histogram at the location specified and scaled to maxX and      *)
(* maxY.                                                                      *)

PROCEDURE showhistogram;

  var
    height , width : integer;
    classlabel : real;
    percent : real;
    unitheight, unitwidth : integer;
    index, barlines : integer;
    maxcount : integer;

{ draws a bar of the histogram at the current position }
PROCEDURE drawbar (num : integer;
                   unitH, unitW,height : integer);
  var savex,savey,color : integer;
 begin
    savex := getx;
    savey := gety;
    if num > 0 then
      bar3D(getx,gety - num * unitH,getx + unitW - x(3),gety,0,true);
   moveto(savex + unitW, savey);
 end;


PROCEDURE addhorizon_axis (xcoord, ycoord, width, high ,linespeed: integer);
   var
     i : integer;
     savex,savey : integer;
     numsec : real;
 begin
  settextjustify(centertext,toptext);
  moveto(xcoord + width, ycoord);
  numsec := high / linespeed;
  for i := 0 to 3 do
    begin
      savex := getx;
      savey := gety;
      linerel(0, y(3));
      moverel(0, y(5));
      outtext(real_to_string(numsec - (numsec / 4) * i));
      moveto(savex,savey);
      moverel(-(width div 4), 0);
    end;
  settextjustify(lefttext,bottomtext);
 end;

begin
 height := y(y_axis_height);                 {scale the height and width to the local }
 width := x(x_axis_width);                   {coordinate system. }
 xcoord := x(xcoord);
 ycoord := y(ycoord);

 moveto(xcoord + width, ycoord);
 lineto(xcoord, ycoord);                     { draw x and y axis }
 lineto(xcoord, ycoord - height);
```

```
  moveto(xcoord + width div 4, ycoord - (height + y(22)));
   outtext(histo.title);
  moveto(xcoord + width div 3, ycoord + y(35));
   outtext(histo.hor_title);                    { Add label and title }
  moveto(xcoord - x(70), ycoord - height div 3);
  outtext('Number');
  moveto(xcoord, ycoord);

   with histo do
    begin
     maxcount := counters[1];
     for index := 2 to noofclasses do
       if counters[index] > maxcount then        { find tallest bar }
         maxcount := counters[index];
      if maxcount > 0 then
        unitheight := trunc(height / maxcount)   { scale to tallest bar }
      else
         unitheight := 0;
     unitwidth := trunc(width / noofclasses);
     for index := 1 to noofclasses do
      drawbar(counters[index], unitheight, unitwidth,height)
    end;

  moveto(xcoord, ycoord - height div 2);
  linerel(x(-3), 0);
  moverel(x(-5), 0);
  settextjustify(righttext,centertext);
  outtext(real_to_string(maxcount DIV 2));       { add scales on X and Y axis }
  moveto(xcoord, ycoord - height);
  linerel(x(-3), 0);
  moverel(x(-5), 0);
  outtext(real_to_string(maxcount));
   settextjustify(lefttext,bottomtext);
  addhorizon_axis(xcoord, ycoord, width, histo.high, speed);

   moveto(xcoord, ycoord + y(65));              { Add mean and std dev at bottom }
   outtext(' Mean = ');
   if histo.tally.noofobs > 0 then
    outtext(real_to_string(histo.tally.integral / histo.tally.noofobs))
   else
    outtext('0.0');
   moverel(x(20), 0);
   outtext(' Sigma = ');
  outtext(real_to_string(TalliedSigmaQ(histo.tally)));
   settextjustify(centertext,bottomtext);
   outtextXY(x(200),y(295),footer);
end;
```

```
{-----------------------------------ShowPieChart-----------------------------------}
(* Displays a pie chart in the middle of the screen and scaled to maxX and      *)
(* maxY.                                                                        *)

PROCEDURE showpiechart;
 var
    total, start, totaldata,data : integer;
    radius : word;
    x,y : integer;
    centX,centY : integer;
    Xasp,Yasp : word;
    space : integer;
   contpercent,fillpercent,datapercent : integer;


 PROCEDURE GetTextCoords(AngleInDegrees, Radius : word; var X, Y : integer);
  { Get the coordinates of text for pie slice labels }
  var
   Radians : real;
 begin
   Radians := AngleInDegrees * Pi / 180;
   X := round(Cos(Radians) * Radius);
   Y := round(Sin(Radians) * Radius);
 end; { GetTextCoords }


 FUNCTION AdjAsp(Value : integer) : integer;
 { Adjust a value for the aspect ratio of the device }
 begin
   AdjAsp := (LongInt(Value) * Xasp) div Yasp;
 end; { AdjAsp }


 FUNCTION aligntext(angle : integer) : boolean;
    var vert , hor : integer;
   begin
    if (angle > 0) and (angle < 180) then
      vert := bottomtext
    else
     vert := toptext;
    if (angle > 90) and (angle < 270) then
     begin
       hor := righttext;
       aligntext := true
     end
     else
     begin
       aligntext := false;
       hor := lefttext;
     end;
    settextjustify(hor,vert);
   end;
```

```
{ total up the data character counts for each logical channel}
FUNCTION adddata(lcnpt : lcnptr): integer;
   var total : integer;
  begin
    total:= 0;
    while lcnpt <> nil do
    begin
      total := total + lcnpt^.count;
      lcnpt := lcnpt^.next;
    end;
   adddata := total;
  end;

begin
 GetAspectRatio(Xasp, Yasp);
 centX := getmaxX div 2;
 centY := getmaxY div 2;

 totaldata := adddata(lcn);
 total := fill + control + totaldata;
 radius := getmaxX div 6;                    { Calculate percentages for fill }
 start := 0;                                 { and control slices.          }
 fillpercent := round((fill/total) * 100);
 fill := round((fill/total) * 360);
 contpercent := round((control/total) * 100);
 control := round((control/total) * 360);

 settextjustify(centertext,toptext);
 outtextXY(centX,3 * textheight('H'),title);  { add title }
 setfillstyle(3,getcolor);

 if fill > 0 then
 begin
  PieSlice(centX,centY,start,start + fill,radius);
  if aligntext(start + fill div 2) then
   space := -textwidth('HH')
  else                                         { draw fill pie slice }
  space := textwidth('HH');
  GetTextCoords(start + fill div 2,radius,x,y);
  outtextXY(centX + x + space,CentY - AdjAsp(y), ('Fill ' + int_to_string(fillpercent) + '%'));
  start := fill;
 end;

 if control > 0 then
 begin
  setfillstyle(6,getcolor);
  PieSlice(centX,centY,start,start + control,radius);
  if aligntext(start + control div 2) then
   space := -textwidth('HH')
  else                                         {draw control pie slice }
  space := textwidth('HH');
  GetTextCoords(start + control div 2,radius,x,y);
  outtextXY(centX + x + space,CentY - AdjAsp(y), ('Control '+ int_to_string(contpercent) + '%'));
  start := start + control;
 end;
```

```
  while (lcn <> nil) do
  begin
   data := round((lcn^.count/total) * 360);
   datapercent := round((lcn^.count/total) * 100);
   if data > 0 then
    begin
      if start + data > 360 then                    { add data pie slices }
      data := 360 - start;                          { one for each logical channel }
      setfillstyle(lcn^.lcnnum mod 10 , getcolor);
      PieSlice(centX,centY,start,start + data ,radius);
      if aligntext(start + (data div 2)) then
       space := -textwidth('HH')
      else
       space := textwidth('HH');
      GetTextCoords(start + data div 2,radius,x,y);
      outtextXY(centX + x + space,CentY - AdjAsp(y),('Data for Lcn '+ int_to_string(lcn^.lcnnum)
                                                    + ','+ int_to_string(datapercent) + '%'));
    end;
   start := start + data;
   lcn := lcn^.next;
  end;

  settextjustify(centertext,bottomtext);
  outtextXY(centX,getmaxY - textheight('H'),foot);
 end;

END.
```

```
{ Program to analyse a Print File from the Comstate 1 Datascope. }


program Panalyze;

{$M $4000,0,65536}
USES  CRT,Graph,mygraph,typesunit,Dos,Drivers;

  CONST
   none = -1;                    { do not know what ptype of data being read }
   data = 0;                     { reading data characters }
   fill = 1;                     { reading fill characters }
   Layer_2 = 2;                  { reading layer 2 frame header or control frame }
   Layer_3 = 3;                  { reading layer 3 packet header or control packet }
   up_arrow = #072;              { up arrow key }
   down_arrow = #080;            { down arrow key }

   help_analyse = 1;
   help_pie_graph = 2;
   help_inter_packet = 3;
   help_packet_length = 4;
   help_more = 5;
   help_replications = 6;
   help_dos_shell = 7;

   helpfilename = 'Ana2.hp';     { Name of help file }

  TYPE
   packetype = record
       data : integer;           (* count of layer's datas *)
       rrpack : integer;         (* count of layer's RRs  *)
       rejpack : integer;        (* count of layer's REJs (none for layer 3) *)
       rnrpack : integer;        (* count of layer's RNRs  *)
       other : integer;          (* count of layer's other frames   *)
     end;

   nextptr = ^lcnstruct;
   lcnstruct = record
       lcn_number : integer;     (* logical channel number *)
       L3packets : packetype;    (* count of layer 3 packets for this lcn *)
       charactercount : integer; (* count of characters for this lcn *)
       next : nextptr;           (* ptr to next lcn record *)
     end;

   statstruct = record
       histint : ahistogramtype; (* histogram of inter-data-packet times *)
       histpac : ahistogramtype; (* histogram of data packet lengths *)
       frametype : integer;      (* type of frame currently in *)
       fillcount : integer;      (* count of fill characters *)
       logicalchannel : nextptr; (* pointer to linked kist of LC's *)
       this_lcn : nextptr;       (* ptr to LCN record of current packet *)
       lcn  : integer;           (* LCN of the current packet *)
       L2packets : packetype;    (* count of  layer 2 frame types  *)
       datacount : integer;      (* total of user data chars seen *)
       controlcount : integer;   (* count of control chars *)
       oschars : integer;        (* count of chars since last space) *)
       headcount : integer;      (* count of chars seen in current header *)
       presentlyin : integer;    (* type of data reading at the present  *)
       interpacket : integer;    (* count of inter-data-packet characters *)
       thispacket : integer;     (* count of chars in this packet *)
       thisfill : real;          (* count of chars in this fill section *)
       graph : text;             (* file of statistics for spreedsheet *)
     end;
```

```
  menu_ptr = ^menustruct;
  menustruct = record                (* node in a menu list *)
    text : string;                   (* text to be displayed *)
    next : menu_ptr;                 (* ptr to next node down list *)
    last : menu_ptr;                 (* ptr to next node up the list *)
  end;

  help_array = array [help_analyse..help_dos_shell] of integer;

  VAR
   help_index : help_array;
   help_file : text;

   selection : integer;              (* which item was chosen from main menu *)

   dce, dte : statstruct;            (* records for storing line statistics   *)
                                     (* these are used or added to by most  *)
                                     (* procedures. To clean up code not    *)
                                     (* passed as a parameter to everything *)
                                     (* that uses them.                     *)

   testfile, savefile : text;        (* input and output text files *)

   outputfilename,inputfilename : STRING;   (* text files names *)

   lookup : ARRAY['!'..'~', '!'..'~'] OF integer;
                                          (* mnemonic conversion array *)

   quit : boolean;                        (* exit from program *)
   analyse_file : boolean;                (* flag to indicate if there results to graph *)

   previous_analyse_file : boolean;
   BitMenu , MainMenu , GraphMenu : menu_ptr;  (* menus *)
   prompt : string;                       (* key pronpt *)
   line_speed : integer;                  (* the line speed selected *)
   dterep , dcerep : replication;
   add_to_series : boolean;


{-------------------------------Error_Handler-------------------------------}
(* General purpose error message routine: displays the message, wait for    *)
(* return to continue.                                                      *)

procedure error_handler(message : string);
  begin
    writeln;
    writeln(message);
    writeln('Type RETURN to continue.');
    readln;
  end;
```

```
{-------------------------------RegisterDrivers-------------------------- }
{ register the drivers for the different graphics cards.                  }
Procedure RegisterDrivers;
 begin
  if 0 > RegisterBGIdriver(@CGADriverProc) then
   error_handler('CGA not registered');
  if 0 > RegisterBGIdriver(@EGAVGADriverProc) then
   error_handler('EGA not registered');
  if 0 > RegisterBGIdriver(@HERCDriverProc) then
   error_handler('HERC not registered');
  if 0 > RegisterBGIdriver(@ATTDriverProc) then
   error_handler('ATT not registered');
  if 0 > RegisterBGIdriver(@PC3270DriverProc) then
   error_handler('PC 3270 not registered');
 end;


{-----------------------------Display-----------------------------------}
(* Dispalys the menu passed to it in the linked list at the position     *)
(* indicated by xpos and ypos (expressed in my coordinte system) and with *)
(* the given prompt at the bottom                                        *)

function display(menu : menu_ptr; xpos,ypos : integer; prompt : string):integer;
var
 ptr : menu_ptr;
 textmax : integer;
 pad : integer;
 count : integer;
 grdriver : integer;
 grmode : integer;
 linesp : integer;
 i : integer;
 number : integer;
 command : char;
 no_color, norm_color : integer;
 ErrCode : integer;
 startx,starty,endx,endy : integer;

 procedure do_menu_move(com : char;
                        var current:integer;
                        var curptr : menu_ptr);

  var startx,starty,endx,endy : integer;

  begin
   setcolor(no_color);
   endx := xpos + pad + textwidth(curptr^.text);
   endy :=  ypos + linesp div 2 +(current + 1) * (textheight('H') + linesp);
   starty := ypos + linesp div 2 + (current + 1) * (textheight('H')+ linesp);
   startx := xpos + pad;
   line(startx, starty, endx, endy);
   if com = up_arrow then
     begin
      if current = 0 then
        current := number
      else
        current := current - 1;
      curptr := curptr^.last;
     end;
   if com = down_arrow then
     begin
      if current = number then
```

```
          current := 0
        else
          current := current + 1;
        curptr := curptr^.next;
      end;
    setcolor(norm_color);
    startx := xpos + pad;
    starty := ypos + linesp div 2 + (current + 1) * (textheight('H') + linesp);
    endx := xpos + pad + textwidth(curptr^.text);
    endy :=  ypos + linesp div 2 + (current + 1) * (textheight('H') + linesp);
    line(startx, starty, endx, endy);
  end;

begin
 grdriver := detect;
 initgraph(grdriver,grmode,'');
 ErrCode := GraphResult;
 if ErrCode = GrOk then
   begin
    xpos := round((getmaxX / mymaxX) * xpos);
    ypos := round((getmaxY / mymaxY) * ypos);
    setTextJustify(lefttext,toptext);
    norm_color := getcolor;
    no_color := 0;
    number := 0;
    linesp := 5;
    pad := textwidth('  ');
    textmax := 0;
    ptr := menu;
    textmax := textwidth(ptr^.text);
    ptr := ptr^.next;
    while ptr <> menu do
    begin
     if textwidth (ptr^.text) > textmax then
       textmax := textwidth(ptr^.text);
     ptr := ptr^.next;
     number := number + 1;
    end;
    while keypressed do
     command := readkey;
    endy := ypos + linesp + (number + 1) * (textheight('H') + linesp);
    rectangle(xpos,ypos,(xpos + textmax + 2 * pad), endy);
    for i := 0 to number  do
      begin
        startx := xpos + pad;
        starty :=  ypos + linesp+ i * (textheight('H') + linesp);
        outtextXY(startx,starty,ptr^.text);
        ptr := ptr^.next;
      end;
    count := 0;
    startx := xpos + pad;
    starty :=  ypos + linesp div 2 + textheight('H') + linesp;
    endx := xpos + pad + textwidth(ptr^.text);
    endy := ypos + linesp div 2 + textheight('H') + linesp;
    line( startx, starty, endx, endy);
    startx := getMaxX div 2 - (textwidth(prompt) div 2 + pad);
    starty := getMaxY - (textheight('H') + 2 * linesp);
    endx := getmaxX div 2 + textwidth(prompt) div 2 + pad;
    endy := getmaxY;
    rectangle(startx, starty, endx, endy);
    SetTextJustify(centertext,centertext);
    outtextxy(getMaxX div 2, getmaxY - (textheight('H') div 2 + linesp),prompt);
    command := 'a';
```

```
    while ord(command) <> 13  do
      begin
       command := readkey;
       if (command = up_arrow) or (command = down_arrow) then
         do_menu_move(command,count,ptr);
       if ord(command) = 13 then
         display := count;
      end;
    closegraph
  end
 else
  begin
   Error_Handler(concat('Graphics Error : ',GraphErrorMsg(ErrCode)));
    halt;
  end;
end;


{-------------------------------Convert-------------------------}
(* Function that takes two characters as input and returns an integer.  *)
(* Treats the two characters as hexidecimal digits and returns an       *)
(* integer containing their base ten value.                             *)

FUNCTION convert (ch : char) : integer;
   var
     i : integer;
 begin
  if (ch >= 'A') then
   i := ord(ch) - ord('A') + 10
  else
   i := ord(ch) - ord('0');
  convert := i;
 end;


{-----------------------------TableLookUp-----------------------------------}
(* Function takes two characters of a numonic and returns the ASCII value    *)
(* of that numonic.                                                          *)

FUNCTION tablelookup (secch, firstch : char) : integer;
   var
     value : integer;
 begin
  value := lookup[firstch,secch];
  if (value <> -1) then
    tablelookup := value
  else
    tablelookup := convert(firstch) * 16 + convert(secch);
 end;


{---------------------------------CreateLookup-------------------------}
(* Initialize the array lookup to contain all -1 to start with (error value  *)
(* if returned) and then sets the values of the numonic pairs which exist.   *)

PROCEDURE createlookup;
  var
   index1,index2 : char;
  begin
   for index1 := '!' to '~' do
     for index2 := '!' to '~' do
       lookup[index1,index2] := -1;
```

```
    lookup['n', 'u'] := 0;
    lookup['s', 'h'] := 1;
    lookup['s', 'x'] := 2;
    lookup['e', 'x'] := 3;
    lookup['e', 't'] := 4;
    lookup['e', 'q'] := 5;
    lookup['a', 'k'] := 6;
    lookup['b', 'l'] := 7;
    lookup['b', 's'] := 8;
    lookup['h', 't'] := 9;
    lookup['l', 'f'] := 10;
    lookup['v', 't'] := 11;
    lookup['f', 'f'] := 12;
    lookup['c', 'r'] := 13;
    lookup['s', 'o'] := 14;
    lookup['s', 'i'] := 15;
    lookup['d', 'l'] := 16;
    lookup['d', '1'] := -1;
    lookup['d', '2'] := -1;
    lookup['d', '3'] := -1;
    lookup['d', '4'] := -1;
    lookup['n', 'k'] := 21;
    lookup['s', 'y'] := 22;
    lookup['e', 'b'] := 23;
    lookup['c', 'n'] := 24;
    lookup['e', 'm'] := 25;
    lookup['s', 'b'] := 26;
    lookup['e', 'c'] := 27;
    lookup['f', 's'] := 28;
    lookup['g', 's'] := 29;
    lookup['r', 's'] := 30;
    lookup['u', 's'] := 31;
    lookup['s', 'p'] := 32;
  end;


{-----------------------------Standard_Format_File-----------------------}
(* Checks the three strings against the three strings expected at the     *)
(* start of a standard format file as obtained from the comstate with a  *)
(* print of the data buffer.                                             *)

FUNCTION standard_format_file(line1,line2,line3 : string):boolean;
  begin
    standard_format_file := false;
    if (copy(line3,1,5) = 'ASCII') then
      standard_format_file := true
  end;


{-----------------------------Init_help----------------------------------}
(* This procedure does the opens the help file and does the initialising  *)
(* of the help_index array.                                              *)

PROCEDURE init_help (var help_file : text;
                     var help_index : help_array);
  var
    index,linecount : integer;
    ch : char;
  begin
    for index := help_analyse to help_dos_shell do
      help_index[index] := 0;
    assign(help_file,helpfilename);
    {$I-}
```

```
  reset(help_file);
  {$I+}
  if IOResult <> 0 then
   error_handler('Help File Not Found !!')
  else
    begin
     index := 1;
     linecount := 1;
    while ((not(eof(help_file))) and (index <= help_dos_shell)) do
      begin
       if not(eoln(help_file)) then
        begin
         read(help_file,ch);
         if('0' <= ch) and (ch <= '9') then
           begin
            help_index[index] := linecount;
            index := index + 1;
           end;
        end;
        readln(help_file);
         linecount := linecount + 1;
     end;
  end;
end;



{-------------------------------SetUpFiles-------------------------------}
(* Prompt for and reads in the names of the input and output files      *)
(* The files are checked for existence and format at each stage before  *)
(* continuing with the next. If the procedure succeds it returnds the   *)
(* set file variables and a true value. If it fails it displays an      *)
(* appropriate error message, closes any open files and returns false.  *)

FUNCTION setupfiles (var inputfilename,outputfilename : string) : boolean;
 var
    done : boolean;
     line1,line2,line3 : string;
     path : string;
     Err : integer;
    outputdtegraph,outputdcegraph: string;
 begin
  done := false;
  GetDir(0,path);
  if IOResult <> 0 then
    path := '??';
  writeln('Enter the file you want the analyzer to use as input.');
  write(path,'> ');
  readln(inputfilename);
  if length(inputfilename) = 0 then
   error_handler('No File Name Entered !!')
  else
  begin
    assign(testfile,inputfilename);
    {$I-}
    reset(testfile);
    {$I+}
    if IOResult <> 0 then
      error_handler('File Not Found !!')
    else
    begin
     readln(testfile,line1);
     readln(testfile,line2);
```

```
      readln(testfile,line3);
      if standard_format_file(line1,line2,line3) then
       begin
        writeln('Enter the file you want the analyser to use to output results to');
        write(path,'> ');
       readln(outputfilename);
       if length(outputfilename) = 0 then
       begin
        close(testfile);
       error_handler('No File Name Entered !!')
       end
       else
       begin
        assign(savefile,outputfilename);
        {$I-}
       rewrite(savefile);
        {$I+}
        if IOResult <> 0 then
        begin
         close(testfile);
        error_handler('File error trying to open result file for writing !!')
        end
        else
        begin
        outputdtegraph :=concat(outputfilename,'dte.gr');
        outputdcegraph := concat(outputfilename,'dce.gr');
        assign(dte.graph,outputdtegraph);
        assign(dce.graph,outputdcegraph);
         {$I-}
        rewrite(dte.graph);
         {$I+}
         if IOResult <> 0 then
         begin
          close(savefile);
          close(testfile);
         error_handler('File Error with dte graph file !!')
         end
         else
         begin
          {$I-}
         rewrite(dce.graph);
          {$I+}
          if IOResult <> 0 then
          error_handler('File Error with dce graph file !!')
          else
           done := true;
        end
       end
      end
     else
     begin
      close(testfile);
     error_handler(' File not in standard format for analyzer !!');
     end;
    end;
  end;
  setupfiles := done;
end;
```

```
{-------------------------------GetBitRate-----------------------------}
(* Displays the bit rate menu and returns the bit rate selected.       *)
(* If "Do Not Know" is selected a bit rate of 8/sec is returned this   *)
(* gives the histogram a scale of 1 char/sec.                          *)

FUNCTION getbitrate : integer;
 var selection : integer;
 begin
  selection := display(BitMenu,130,100,prompt);
  case selection of
   0 : getbitrate := 1200;
   1 : getbitrate := 2400;
   2 : getbitrate := 4800;
   3 : getbitrate := 9600;
   4 : getbitrate := 8;
  end;
 end;



{----------------------Init_packet_counts-----------------------}

PROCEDURE init_packet_counts (VAR thisone : packetype);
 begin
  with thisone do
   begin
    data := 0;
    rrpack := 0;
    rnrpack := 0;
    rejpack := 0;
    other := 0;
   end;
 end;


{---------------------------------Setup-------------------------------}
(* Initializes the statstruct passed to it to the initial values.     *)

PROCEDURE setup (var present : statstruct);
 begin
  with present do
   begin
    frametype := -1;
    fillcount := 0;
    new(logicalchannel);
    logicalchannel^.next := NIL;
    logicalchannel^.lcn_number := -1;
    logicalchannel^.charactercount := 0;
    init_packet_counts(logicalchannel^.L3packets);
    this_lcn := NIL;
    lcn := -1;
    init_packet_counts(L2packets);
    controlcount := 0;
    oschars := 0;
    headcount := 0;
    presentlyin := none;
    interpacket := 0;
    thispacket := 0;
    thisfill := 0;
   end;
 end;
```

```
{-----------------------------GatheringStats-----------------------------}
(* Gathers the statistics of character counts from each of the logical   *)
(* channels and returns them in chcount.                                 *)

PROCEDURE gatheringstats (var present : statstruct;
                          var chcount : integer);
  var
    this : nextptr;
 begin
  this := present.logicalchannel;
  chcount := 0;
  while (this^.next <> NIL) do
    begin
     chcount := this^.charactercount + chcount;
      this := this^.next;
    end;
 end;


{-----------------------------------WriteOut-----------------------------------}
(* Writes out the statistics gathered from analyzing the input file to the    *)
(* output file selected.                                                      *)

 PROCEDURE writeout (var present : statstruct);
  const align = 45;
  var
   percent : real;
   next_lcn : nextptr;
   c : char;

  FUNCTION leftjust(intext : string):string;
   var i : integer;
   begin
     for i := length(intext) to align do
         intext := intext + ' ';
     leftjust := intext;
   end;

 PROCEDURE writeoutlcn (this : nextptr);
  begin
  with this^ do
   begin
   writeln(savefile,leftjust('Logical channel number'),'=':2,lcn_number : 6);
   writeln(savefile,leftjust('Logical channel data character count'),'=':2, charactercount : 6);
   with L3packets do
   begin
   writeln(savefile,leftjust('Logical channel layer 3 data packet count'),'=':2, data:6);
   writeln(savefile,leftjust('Logical channel layer 3 rr packet count'),'=':2, rrpack:6);
   writeln(savefile,leftjust('Logical channel layer 3 rnr packet count'),'=':2, rnrpack:6);
   writeln(savefile,leftjust('Logical channel layer 3 other packet count'),'=':2, other:6);
   end;
  end;
 end;

 begin
  gatheringstats(present, present.datacount);
  with present do
    begin
     writeln(savefile,leftjust('send fill count'),'=':2, fillcount  : 6);
     writeln(savefile,leftjust('send control count'),'=':2, controlcount : 6);
     writeln(savefile,leftjust('send user data count'),'=':2, datacount : 6);
     writeln(savefile,leftjust('send L2 data packet count'),'=':2,L2packets.data : 6);
```

```
      writeln(savefile,leftjust('send L2 rr packet count'),'=':2,L2packets.rrpack : 6);
      writeln(savefile,leftjust('send L2 rnr packet count'),'=':2,L2packets.rnrpack : 6);
      writeln(savefile,leftjust('send L2 rej packet count'),'=':2,L2packets.rejpack : 6);
      writeln(savefile,leftjust('send L2 other packet count'),'=':2,L2packets.other : 6);
      percent := (fillcount / (datacount + controlcount + fillcount)) * 100;
      writeln(savefile,leftjust('Percentage of idle send line time'),'=':2,percent : 6 : 2,'%');
      writeln(savefile);
      next_lcn := logicalchannel;
      while (next_lcn^.next <> NIL) do
        begin
         writeoutlcn(next_lcn);
         next_lcn := next_lcn^.next;
         writeln(savefile);
        end;
    end;
 end;


{-----------------------------------Readin-----------------------------------}
(* This is the only procedure which reads from the input file. (Apart from   *)
(* the strings read to test if the file is in the standard input format, and  *)
(* readln's at the end of each line and removing blank lines between the dual *)
(* format). Called by all other procedures which require characters from the  *)
(* input file.                                                                *)

 PROCEDURE readin (VAR ch : char);
   begin
    read(testfile, ch);
   end;


{-------------------------Find_lcn_Record--------------------------}
(* Searchs the linked list of lcn records for one which has the requested *)
(* lcn number. If one does not exist it creates a new node and adds it to *)
(* the end of the list.                                                   *)

PROCEDURE find_lcn_record (head : nextptr;
                var current : nextptr;
                this_lcn : integer);
   var
     found : boolean;
  begin
   found := false;
   current := head;
   while ((current^.next <> NIL) and (not found)) do
     begin
      if (current^.lcn_number = this_lcn) then
        found := true
       else
        current := current^.next;
     end;
   if (not found) then
     begin
      new(current^.next);
      current^.next^.next := NIL;
      current^.lcn_number := this_lcn;
      current^.charactercount := 0;
      init_packet_counts(current^.L3packets);
     end;
  end;
```

```
{-----------------------------Layer2-------------------------------}
(* Reads in and analyzes the layer two header of the packet. If the frame      *)
(* type indicates it contains data for layer 3 the procedure finishes. If      *)
(* it is a layer 2 frame only then it is all read in. The appropriate          *)
(* statistics counters are incremented.                                        *)

PROCEDURE layer2 (VAR thisone : statstruct);
  var
    finished : boolean;
    ch, lastch : char;
    lcn : integer;
 begin
  finished := false;
  lastch := ' ';
  with thisone do
    begin
     while ((not (eoln(testfile))) and (not finished)) do
       begin
        readin(ch);
         if (ch <> ' ') then
           oschars := oschars + 1;
        if ((ch = ' ') or (oschars = 2)) then
          begin
         controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
         headcount := headcount + ((oschars div 2) + (oschars mod 2));
          oschars := 0;
          interpacket := interpacket + 1;

           {second octet layer 2 extract type of frame}
         if ((headcount = 2) and (frametype = -1)) then
            begin
              if (ch <> ' ') then
               frametype := tablelookup(ch, lastch)
              else
               frametype := ord(lastch);
             if ((frametype mod 2) = 0) then  {if info frame}
               begin
                presentlyin := layer_3;
               L2packets.data := L2packets.data + 1;
                finished := true
              end
             else
               begin
                frametype := frametype mod 32;
                 if frametype in [1,5,17] then
                   case frametype of
                      17 :
                         L2packets.rejpack := L2packets.rejpack + 1;
                       5 :
                         L2packets.rnrpack := L2packets.rnrpack + 1;
                       1 :
                         L2packets.rrpack := L2packets.rrpack + 1
                   end
                 else
                   L2packets.other := L2packets.other + 1;
               end;
             frametype := 0;
            end;
       end;
      if ((lastch = ':') and (ch = 'G')) then   {if layer 2 trailer}
        begin
         controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
```

```
          interpacket := interpacket + ((oschars div 2) + (oschars mod 2));
           oschars := 0;
           presentlyin := fill;
           finished := true;
           headcount := 0;
           frametype := -1;
          end;
        lastch := ch;
       end;
    end;
 end;


{------------------------------Layer3-----------------------------------}
(* Reads in and analyzes the layer three header of the packet. If the packet     *)
(* type indicates that it is a data packet then the procedure finishes           *)
(* otherwise all the packet is read in as layer 3 and the procedure finishes     *)
(* The fourth numonic is the logical channel number and the fifth is the         *)
(* packet type. The appropriate statistical counters are updated.                *)

PROCEDURE layer3 (VAR thisone : statstruct);
   var
    finished : boolean;
    ch, lastch : char;
  begin
   finished := false;
   lastch := ' ';
   with thisone do
     begin
      while ((not (eoln(testfile))) and (not finished)) do
        begin
         readin(ch);
          if (ch <> ' ') then
            oschars := oschars + 1;
         if ((ch = ' ') or (oschars = 2)) then
           begin
            controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
            headcount := headcount + ((oschars div 2) + (oschars mod 2));
            interpacket := interpacket + ((oschars div 2) + (oschars mod 2));
             oschars := 0;

            {first octet layer 3 header extract top part of LCN}
            if ((headcount = 3) and (lcn = -1)) then
               begin
                 if (ch <> ' ') then
                   lcn := tablelookup(ch, lastch) mod 16
                 else
                   lcn := ord(lastch) mod 16;
               end;

            {second octet layer 3 header extract bottom part LCN}
            if ((headcount = 4) and (this_lcn = NIL)) then
               begin
                 if (ch <> ' ') then
                   lcn := lcn * 256  + tablelookup(ch, lastch)
                 else
                   lcn := lcn * 256 + ord(lastch);
                find_lcn_record(logicalchannel, this_lcn, lcn);
               end;
```

```
          {third octet layer 3 header extract type of packet}
         if ((headcount = 5) and (frametype = 0)) then  {third octet}
            begin
             if (ch <> ' ') then
              frametype := tablelookup(ch, lastch)
             else
              frametype := ord(lastch);
            if ((frametype mod 2) = 0) then    {if data packet}
             begin
              presentlyin := data;
              headcount := 0;
              oschars := 0;
              finished := true;
             this_lcn^.L3packets.data := this_lcn^.L3packets.data + 1;
            end
            else
            begin
             frametype := frametype mod 32;
             with this_lcn^ do
               begin
                if frametype in [1,5] then
                 case frametype of
                    5 :
                     L3packets.rnrpack := L3packets.rnrpack + 1;
                    1 :
                     L3packets.rrpack := L3packets.rrpack + 1
                  end
                else
                  L3packets.other := L3packets.other + 1;
              end; { with}
             end;
            end;
        end;
       if ((lastch = ':') and (ch = 'G')) then     {layer 2 trailer}
          begin
          controlcount := controlcount + ((oschars div 2) + (oschars mod 2));
          interpacket := interpacket + ((oschars div 2) + (oschars mod 2));
           oschars := 0;
           presentlyin := fill;
           finished := true;
           headcount := 0;
           frametype := -1;
           this_lcn := NIL;
           lcn := -1;
         end;
        lastch := ch;
       end;
    end;
end;
```

```
{-------------------------------ReadData-------------------------------}
(* Reads in the rerst of a data packet after layer2 and layer3 have removed     *)
(* the frame and packet headers. The appropriate statistical counter are        *)
(* incremented. The number of characters in the data packet and the timke       *)
(* since the end of the last data packet (to the start of this one) are         *)
(* written out to the file d_e.gr with a tab between them.                      *)

PROCEDURE readdata (var thisone : statstruct; char_rate : integer);
  var
   finished : boolean;
   ch, lastch : char;
   numchars : integer;
 begin
  finished := false;
  with thisone do
   begin
    while ((not (eoln(testfile))) and (not finished)) do
      begin
       readin(ch);
        if (ch <> ' ') then
         oschars := oschars + 1;
       if (ch = ' ') then
         begin
          numchars := (oschars div 2) + (oschars mod 2);
         this_lcn^.charactercount := this_lcn^.charactercount + numchars;
          thispacket := thispacket + numchars;
          oschars := 0;
         end;
       if ((lastch = ':') and (ch = 'G')) then
         begin
          if (oschars = 2) then
            begin
             controlcount := controlcount + 2;  { add layer 2 trailer }
            this_lcn^.charactercount := This_lcn^.charactercount - 1;          { to remove layer 2 trailer}
             thispacket := thispacket - 1 { to remove layer 2 trailer }
            end
          else
           controlcount := controlcount +(oschars div 2)+(oschars mod 2);
          oschars := 0;
          presentlyin := fill;
          finished := true;
          frametype := -1;
          this_lcn := NIL;
          lcn := -1;
          updatehist(histint,interpacket);
          updatehist(histpac,thispacket);
         writeln(graph, interpacket/char_rate,chr(9), thispacket);
          thispacket := 0;
          interpacket := 0;
         end;
        lastch := ch;
      end;
   end;
end;
```

```
{-------------------------ReadFill-------------------------}
(* Reads in and count '.' until a character which is not a '.' is      *)
(* encountered.                                                        *)

PROCEDURE readfill (var thisone : statstruct);
   var
    finished : boolean;
    ch, lastch : char;
    i : integer;
 begin
  ch := ' ';
  lastch := ' ';
  finished := false;
  with thisone do
   begin
    while ((not (eoln(testfile))) and (not finished)) do
      begin
       readin(ch);
       if (ch = '.') then
           thisfill := thisfill + 1
       else
         begin
          if (ch <> ' ') then
           oschars := oschars + 1;
          if ((lastch <> ' ') and (lastch <> '.')) then
           begin
            finished := true;
            interpacket := interpacket + round(thisfill / 3);
            fillcount := fillcount + round(thisfill / 3);
             thisfill := 0;
            presentlyin := layer_2;
           end;
         end;
       lastch := ch;
      end;
   end;
 end;


{-------------------------ReadStart-------------------------}
(* Reads in until characters util it can determine where abouts it is.  *)
(* This is when fill or an end of packet is encountered.                *)

PROCEDURE readstart (var thisone : statstruct);
   var
    finished : boolean;
    ch, lastch : char;
    i : integer;
 begin
  ch := ' ';
  lastch := ' ';
  finished := false;
  with thisone do
   begin
    while ((not (eoln(testfile))) and (not finished)) do
      begin
       readin(ch);
       if (ch = '.') and (lastch = '.') then
        begin
          finished := true;
          presentlyin := fill
       end
```

```
        else if (ch = 'G') and (lastch = ':') then
          begin
           finished := true;
           presentlyin := fill
          end;
         lastch := ch;
        end;
    end;
 end;


{-------------------------------CountUp-------------------------------}
(* Loops around selecting the next procedure required (selected on the value    *)
(* of presentlyin). This countinues until the end of a line is reached.        *)

PROCEDURE countup (var present : statstruct;char_rate : integer);
 begin
  with present do
   begin
    while not (eoln(testfile)) do
      begin
       case presentlyin of
         none :
          readstart(present);
          fill :
          readfill(present);
         layer_2 :
          layer2(present);
         data :
         readdata(present,char_rate);
         layer_3 :
          layer3(present);
       end;
      end;
    readln(testfile);
   end;
 end;


{-----------------------------Analyse-----------------------------------}
(* Calls setupfiles to obtain the file names, and if they are valid it         *)
(* initialises the statistical counters and histograms then ananlyzes the      *)
(* file. The results are writen out to the requested file. Returns             *)
(* success = true if there are no errors in the analyzing                      *)

PROCEDURE analyse(var success : boolean; var char_rate : integer);
 var bitrate : integer;
  begin
   if setupfiles(inputfilename,outputfilename) then
     begin
      bitrate := getbitrate;
      char_rate := bitrate div 8;
      setup(dte);                                    (* Initialize the statistics counters *)
      writeln(dte.graph, 'Interpacket', chr(9), 'Packet Length');
      setup(dce);
      writeln(dce.graph, 'Interpacket', chr(9), 'Packet Length');
      writeln(savefile, 'Results of analyzing ', inputfilename);
      inithistogram(dte.histint,0,600,25, 'Histogram of DTE interpacket times ','DTE interpacket time
                                                                          in seconds ');
      inithistogram(dce.histint,0,600,25,'Histogram of DCE interpacket times','DCE interpacket time
                                                                          in seconds ');
      inithistogram(dte.histpac,0,256,25,'Histogram of DTE packetlengths','DTE packet length in
                                                                          chars');
```

```
      inithistogram(dce.histpac,0,256,25,'Histogram of DCE packetlengths','DCE packet length in
                                                                                    chars');

      while (not (eof(testfile))) do
        begin
         countup(dte,char_rate);          (* one line for each direction *)
         countup(dce,char_rate);
         readln(testfile);                (* remove blank line between dual lines *)
        end;

      writeln(savefile, 'DTE Results');   (* Print out the results *)
      writeout(dte);
      writeln(savefile);
      writeln(savefile, 'DCE Results');
      writeout(dce);
      close(testfile);
      close(savefile);
      writeln(dte.graph);
      writeln(dce.graph);
      close(dte.graph);
      close(dce.graph);
      success := true
    end
  end;


{-----------------------------------Display_graph-------------------------------}
(* Initialises graphics mode and then calls showhistogram to draw it on       *)
(* screen. It then waits fro return key, closes graphics mode and returns.    *)

PROCEDURE display_graph(histo : ahistogramtype;
                        linespeed : integer; footer : string);
  var grDriver : integer;
     grMode : integer;
     ErrCode : integer;

  begin
    grDriver:= Detect;
    InitGraph(grDriver,grMode,'');
    ErrCode := GraphResult;
    if ErrCode = grOk then
      begin
        showhistogram(histo,100,200,linespeed,footer);
        readln;
        closeGraph;
      end
    else
     error_handler(concat('Graphics Error: ',GraphErrorMsg(ErrCode)));
  end;


{-----------------------------------Display_pie---------------------------------}
(* Initialises graphics mode and then calls showpiechart to draw it on        *)
(* screen. It then waits for return key, closes graphics mode and returns.    *)

PROCEDURE display_pie(fill,control : integer;
                      lcn : lcnptr;
                      title : string;
                      footer : string);
  var grDriver : integer;
     grMode : integer;
     ErrCode : integer;
```

```
PROCEDURE freeLCN (lcn : lcnptr);
  var trailer : lcnptr;
  begin
   while lcn <> nil do
    .begin
      trailer := lcn;
      lcn := lcn^.next;
      dispose(trailer);
    end;
  end;

begin
  grDriver:= Detect;
  InitGraph(grDriver,grMode,'');
  ErrCode := GraphResult;
  if ErrCode = grOk then
    begin
      showpiechart(fill,control,lcn,title,footer);
      readln;
      closeGraph;
      freelcn(lcn);
    end
   else
    error_handler(concat('Graphics Error: ',GraphErrorMsg(ErrCode)));
end;


{-------------------------------DataLCN ------------------------------}
(* Extracts the infomation from the linked list of LCN's and          *)
(* constructs a list of LC numbers and data counts.                   *)

FUNCTION datalcn (this : statstruct) : lcnptr;
 var
  lcn, head,trailer :lcnptr;
  current : nextptr;
 begin
   with this do
  current := logicalchannel;
  new(lcn);
  head := lcn;
  trailer := lcn;
  while current^.next <> nil do
   begin
    lcn^.lcnnum := current^.lcn_number;
    lcn^.count := current^.charactercount;
    current := current^.next;
    new(lcn^.next);
     trailer := lcn;
    lcn := lcn^.next;
  end;
  dispose(lcn);
  trailer^.next := nil;
  datalcn := head;
 end;
```

```
{-----------------------------Select_graph-------------------------}
(* If there are results to graph the graph menu is displayed and      *)
(* display_graph or display_pie called depending on the selection.     *)

PROCEDURE select_graph(analysed : boolean;linespeed : integer);
 var selection : char;
    done, quit : boolean;
 begin
  if analysed then
   begin
    done := false;
    quit := false;
    repeat
     case display(GraphMenu,130,100,prompt) of
      0 : display_graph(dte.histint,linespeed,dtefoot);
      1 : display_graph(dce.histint,linespeed,dcefoot);
      2 : display_graph(dte.histpac,1,dtefoot);
      3 : display_graph(dce.histpac,1,dcefoot);
      4 : display_pie(dte.fillcount,dte.controlcount,datalcn(dte),
                 dtePiehead,dtefoot);
      5 : display_pie(dce.fillcount,dce.controlcount,datalcn(dce),
                 dcePiehead,dcefoot);
       6 : quit := true;
     end
    until quit;
  end
  else
   error_handler(' No results to graph yet. Try analyzing a file.');
 end;

{-----------------------------Display_help-------------------------}
(* This procedure contains all the procedure for handling the help system.  *)
(* The help file is opened by init_help. Uses the standard menus with a     *)
(* different prompt.                                                        *)

PROCEDURE display_help;
 var
   help_prompt : string;
   quit : boolean;

 PROCEDURE help(which_one : integer);
   var i : integer;
      help_line : string;
   begin
    reset(help_file);
    for i := 1 to help_index[which_one] - 1  do
     readln(help_file);
    while (not(eof(help_file))) and (not(eoln(help_file))) do
    begin
     readln(help_file,help_line);
      writeln(help_line);
    end;
    readln;
   end;
```

```
 PROCEDURE help_select_graph;
  var
   quit : boolean;
  begin
   quit := false;
   repeat
    case display(GraphMenu,130,100,help_prompt) of
      0,1 : help(help_inter_packet);
      2,3 : help(help_packet_length);
      4,5 : help(help_pie_graph);
      6 : quit := true;
     end
    until quit;
  end;


begin
 if help_index[1] < 1 then
  error_handler('Help file was not found.Check it is same directory as program.')
 else
   begin
    quit := false;
   help_prompt := 'Chose Item Help is required on. Quit to go back to Program';
    repeat
     case Display(MainMenu,130,100,Help_prompt) of
        0 : ;
        1 : help(help_analyse);
        2 : help(help_replications);
        3 : help_select_graph;
        4 : help(help_more);
        5 : help(help_Dos_shell);
        6 : quit := true;
       end;
     until quit;
   end;
end;



{-----------------------------More-------------------------------}
(* More (eg like unix more) displays a file 20 lines at a time.          *)
(* Return between pages.If it is passed a filename it doesn't prompt the *)
(* user for one.                                                         *)

PROCEDURE more(filename : string);
 const page = 20;
 var
  count : integer;
  textfile : text;
  ch : char;
 begin
  if length(filename) = 0 then
    begin
     writeln('Name of file to be displayed ?');
      write('> ? ');
     readln(filename);
    end;
  if length(filename) > 0 then
    begin
     assign(textfile,filename);
     {$I-}
     reset(textfile);
```

```
      {$I+}
      if IOResult <> 0 then
       error_handler(concat(' Could not open file: ',filename))
      else
      begin
       repeat
         count := 0;
        while ((not(eof(textfile))) and (count < page)) do
         begin
          while not(eoln(textfile)) do
           begin
             read(textfile,ch);
              write(ch);
           end;
          readln(textfile);
           writeln;
           count := count + 1;
         end;
         writeln;
        writeln(' type return to continue');
         writeln;
        readln;
       until(eof(textfile));
       close(textfile)
     end
    end
    else
     error_handler('No File Name Entered !!');
end;


{-------------------------------SetUp_Bit_Menu-------------------------------}
(* Sets up a doublely linked list with the appropriate text values for the  *)
(* bit rate menu.                                                           *)

PROCEDURE setup_bit_menu(var menu : menu_ptr);

 var
  ptr : menu_ptr;
  i : integer;
 begin
  new(ptr);
  menu := ptr;
  for i := 1 to 4 do
    begin
     new(ptr^.next);
     ptr^.next^.last := ptr;
     ptr := ptr^.next;
    end;
  ptr^.next := menu;
  menu^.last := ptr;
  ptr := menu;
  ptr^.text := '1200 bits/sec.';
  ptr := ptr^.next;
  ptr^.text := '2400 bits/sec.';
  ptr := ptr^.next;
  ptr^.text := '4800 bits/sec.';
  ptr := ptr^.next;
  ptr^.text := '9600 bits/sec.';
  ptr := ptr^.next;
  ptr^.text := 'Do not Know.';
  ptr := menu;
 end;
```

```
{-------------------------------SetUp_Graph_Menu-----------------------------}
(* Sets up a doublely linked list with the appropriate text values for the  *)
(* graph selection menu.                                                    *)

PROCEDURE setup_graph_menu(var menu : menu_ptr);

 var
  ptr : menu_ptr;
  i : integer;
 begin
  new(ptr);
  menu := ptr;
  for i := 1 to 6 do
   begin
    new(ptr^.next);
    ptr^.next^.last := ptr;
    ptr := ptr^.next;
   end;
  ptr^.next := menu;
  menu^.last := ptr;
  ptr := menu;
  ptr^.text := 'DTE Inter Packet Times.';
  ptr := ptr^.next;
  ptr^.text := 'DCE Inter Packet Times.';
  ptr := ptr^.next;
  ptr^.text := 'DTE Data Packet Lengths.';
  ptr := ptr^.next;
  ptr^.text := 'DCE Data Packet Lengths.';
  ptr := ptr^.next;
  ptr^.text := 'DTE Pie Chart';
  ptr := ptr^.next;
  ptr^.text := 'DCE Pie Chart';
  ptr := ptr^.next;
  ptr^.text := 'Quit.';
  ptr := menu;
 end;


{---------------------------------SetUp_Menu---------------------------------}
(* Sets up a doublely linked list with the appropriate text values for the  *)
(* main selection menu.                                                     *)

PROCEDURE setup_menu(var menu : menu_ptr);

 var
  ptr : menu_ptr;
  i : integer;
 begin
  new(ptr);
  menu := ptr;
  for i := 1 to 6 do
   begin
    new(ptr^.next);
    ptr^.next^.last := ptr;
    ptr := ptr^.next;
   end;
  ptr^.next := menu;
  menu^.last := ptr;
  ptr := menu;
  ptr^.text := 'Help and information.';
  ptr := ptr^.next;
```

```
    ptr^.text := 'Analyse a data file.';
    ptr := ptr^.next;
    ptr^.text := 'Display Replication Res';
    ptr := ptr^.next;
    ptr^.text := 'Graphs.';
    ptr := ptr^.next;
    ptr^.text := 'Display text file.';
    ptr := ptr^.next;
    ptr^.text := 'DOS Shell.';
    ptr := ptr^.next;
    ptr^.text := 'Quit.';
  end;


{-----------------------------Free_Menu-------------------------------}
(* Frees the memory in the heap occupied by the nodes of the menu it is  *)
(* passed.                                                               *)

PROCEDURE free_menu(menu : menu_ptr);
  begin
    menu^.last^.next := nil;
    while menu^.next <> nil do
      begin
        menu := menu^.next;
        dispose(menu^.last);
      end;
    dispose(menu);
  end;


{-------------------------TidyUp--------------------------}
(* Calls free-menu for each of the menus created.        *)

PROCEDURE tidyup;
  begin
    free_menu(MainMenu);
    free_menu(GraphMenu);
    free_menu(BitMenu);
  end;


{---------------------Continue_Series---------------------}
FUNCTION continue_series : boolean;
  var
    ch : char;
  begin
    ch := ' ';
    while (ch <> 'n') and (ch <> 'N') and (ch <> 'y') and (ch <> 'Y') do
      begin
        write(' Is this sample another one in the same series ?? (Y / N).');
        readln(ch);
      end;
    if (ch = 'y') or (ch = 'Y') then
      continue_series := true
    else
      continue_series := false;
  end;
```

```
{-------------------------Tally_Up-------------------------------}
(* Adds the results of the current analysis to the tally of the series to      *)
(* date. If add_to_series is false then it zeros the tallies before starting   *)

PROCEDURE tally_up(add_to_series : boolean; var dterep,dcerep : replication);
 var
  dtetotal, dcetotal : integer;
 begin
  if not(add_to_series) then
    begin
     inittally(dterep.data);
     inittally(dcerep.data);
     inittally(dterep.fill);
     inittally(dcerep.fill);
     inittally(dterep.control);
     inittally(dcerep.control);
    end;
   with dte do
    dtetotal := fillcount + controlcount + datacount;
   with dce do
    dcetotal := fillcount + controlcount + datacount;
   updatetally(dterep.data,dte.datacount/dtetotal);
   updatetally(dcerep.data,dce.datacount/dcetotal);
   updatetally(dterep.control,dte.controlcount/dtetotal);
   updatetally(dcerep.control,dce.controlcount/dcetotal);
   updatetally(dterep.fill,dte.fillcount/dtetotal);
   updatetally(dcerep.fill,dce.fillcount/dcetotal);
 end;


{-------------------------Display_Reps-------------------------------}
(* Displays the results of the analysis of the series of samples.      *)

PROCEDURE display_reps(dterep,dcerep:replication);
 var
  dcemean, dtemean, dcestddev, dtestddev : string;

 PROCEDURE getstats(tally : atallytype; var mean , stddev : string);
  begin
   with tally do
   begin
    mean :=  real_to_string((integral / noofobs) * 100);
    stddev := real_to_string(TalliedSigmaQ(tally)* 100);
   end;
  end;

 begin
  writeln('         Results from analysis of Series');
  writeln;
  writeln('Number of observations = ',dterep.data.noofobs);
  writeln;
  writeln('                          Dte to Dce          Dce to Dte');
  writeln('                        Mean    Std Dev      Mean    Std Dev');
  writeln;
  getstats(dterep.data,dtemean,dtestddev);
  getstats(dcerep.data,dcemean,dcestddev);
  writeln('Percentage Data          ',dtemean,'    ',dtestddev,'       ',dcemean,'    ',dcestddev);
  writeln;
  getstats(dterep.control,dtemean,dtestddev);
  getstats(dcerep.control,dcemean,dcestddev);
  writeln('Percentage Control       ',dtemean,'    ',dtestddev,'       ',dcemean,'    ',dcestddev);
  writeln;
```

```
  getstats(dterep.fill,dtemean,dtestddev);
  getstats(dcerep.fill,dcemean,dcestddev);
  writeln('Percentage Fill ',dtemean,' ',dtestddev,' ',dcemean,' ',dcestddev);
  readln;
 end;

{-----------------------------------Main-----------------------------------}
 begin
  init_help(help_file,help_index);
  quit := false;
  RegisterDrivers;
  prompt := 'Main Menu '#25#24' to move ,'#17' to select';
  analyse_file := false;
  previous_analyse_file := false;
  createlookup;
  setup_Bit_menu(BitMenu);
  setup_Graph_menu(GraphMenu);
  setup_menu(MainMenu);
  repeat
    selection := Display(MainMenu,130,100,prompt);
    case selection of

      0 : display_help;

      1 : begin                                          { Analyse }
          previous_analyse_file := analyse_file;
          analyse_file := false;
          analyse(analyse_file,line_speed);
          if analyse_file then
            begin
             if previous_analyse_file then
              previous_analyse_file := continue_series
             else
              previous_analyse_file := false;
            tally_up(previous_analyse_file,dterep,dcerep);
             more(outputfilename);
            end;
          end;

      2 : if previous_analyse_file then                  { Show replication Results }
           display_reps(dterep,dcerep)
          else
           error_handler('No Results to display yet');

      3 : select_graph(analyse_file,line_speed);  { Graphs }

      4 : more('');                                      { Display text file }

      5 : begin                                          { DOS Shell }
           Writeln(' Type EXIT to return to program. ');
           Exec(GetEnv('COMSPEC'),'');
          end;

      6 : quit := true;                                  { Quit }

    end;
  until quit;
  tidyup;
 end.
```